Universität Dortmund
Fachbereich Informatik
Lehrstuhl für Algorithm Engineering - LSXI
PG487

Bericht zum Sommersemester 2006 und Wintersemester 2006/2007

Endbericht PG487

Methoden der Computational Intelligence in der Bioinformatik

André Barthelmes
Frank Behler
Sven Dunkel
Klaus Gerstl
Thorsten Glebocki
Gero Greiner
Andreas Hörsken
David Kampert
David Schichowski
Andreas Schmutzer
Holger Schneider
Akin Sencer

Betreuer
Prof. Dr. Thomas Bartz-Beielstein
Dipl.-Inf. Nicola Beume
Dipl.-Inf. Boris Naujoks

19. April 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Baum-Welch Definitionen

$L(\theta|X)$ Likelihood-Funktion, Seite 16

$\mathbb{X}$ Die Menge aller Ereignisse. Es gilt also $X \in \mathbb{X}$, Seite 16

Θ Die Menge aller Konfigurationen oder Parametereinstellungen, Seite 16

θ Eine Konfiguration oder Parametereinstellung. Es gilt also $\theta \in \Theta$, Seite 16

$\alpha_i(k)$ Wahrscheinlichkeit sich zum Zeitpunkt k in Zustand s_i zu befinden, Seite 17

$\beta_i(k)$ Wahrscheinlichkeit die Sequenz $x_{k+1} \dots x_{|X|}$ zu emittieren unter der Voraussetzung, sich zum Zeitpunkt k in Zustand s_i zu befinden, Seite 17

$P(X|\theta)$ Wahrscheinlichkeit die Sequenz X unter der Konfiguration θ zu emittieren, Seite 17

$\gamma_i(k)$ Wahrscheinlichkeit, unter der aktuellen Konfiguration θ und bei der zu erzeugenden Sequenz X, zum Zeitpunkt k im Zustand s_i des HMM zu sein, Seite 18

$\xi_{i,j}(k)$ Wahrscheinlichkeit, unter der aktuellen Konfiguration θ und der zu erzeugenden Sequenz X, zum Zeitpunkt k in Zustand s_i zu sein und direkt in Zustand s_j über zu gehen, Seite 18

EA Definitionen

λ Anzahl erzeugter Individuen pro Generation, Seite 22

$\mathbb{P}_o$ Population neu erzeugter Individuen, Seite 22

$\mathbb{P}_p$ Population, aus der neue Individuen erzeugt werden, Seite 22

μ Anzahl der Individuen in der Population, Seite 22

ρ Anzahl der Individuen, die für das Erzeugen eines neuen Individuums benutzt werden, Seite 22

κ Lebensdauer eines Individuums (in Generationen), Seite 23

$\vec{s_i}$ Strategieparameter des i-ten Individuums einer Population, Seite 23

$\vec{y_i}$ Objektparameter des i-ten Individuums einer Population, Seite 23

a_i Repräsentation des i-ten Individuums einer Population, Seite 23

$F(\vec{y_i})$ Fitnesswert, der sich aus den Objektparametern $\vec{y_i}$ des i-ten Individuums einer Population berechnet, Seite 23

σ Mutationsstärke, Seite 25

φ Fortschrittsrate, Seite 25

I Wahrscheinlichkeit der Erzeugung eines verbesserten Individuums, Seite 25

HMM Definitionen

HMM Ein Hidden Markov Modell (HMM) ist ein Fünf-Tupel (S, Σ, T, Π, E), Seite 12

Π Startverteilung eines HMM, Seite 13

π_i Startwahrscheinlichkeit für Zustand s_i, Seite 13

Σ Emissionsalphabet eines HMM, Seite 13

σ_j Das j-te Symbol aus einem Emissionsalphabet, Seite 13

E Emissionsmatrix eines HMM, Seite 13

e_{s_i,a_j} Wahrscheinlichkeit, dass in Zustand s_i das Symbol a_j emittiert wird, Seite 13

S Zustandsmenge eines HMM, Seite 13

s_i Der i-te Zustand $s_i \in S$, Seite 13

T Transtionsmatrix eines HMM, Seite 13

$t_{i,j}$ Wahrscheinlichkeit für die Transition von Zustand s_i in Zustand s_j, Seite 13

$t_{\vec{a}}$ Pfadwahrscheinlichkeit eines als Pfad interpretierten Vektors $\vec{a}$, Seite 39

S_{N_i} Menge der Nachfolgerzustände des Zustands i, Seite 48

S_N Menge aller Mengen S_{N_i}, Seite 48

S_{V_i} Menge der Vorgängerzustände des Zustands i, Seite 48

S_V Menge aller Mengen S_{V_i}, Seite 48

Strukturvorhersage Definitionen

PSP Protein Structure Prediction, Seite 91

a Häufigkeit des Auftretens eines Motivs, Seite 95

M_i Menge der Zustände von Elternindividuum i, Seite 96

Viterbi Definitionen

$\delta_k(i)$ Maximale lokale Pfadwahrscheinlichkeit für Zeitpunkt k an Zustand s_i, Seite 15

$\phi_k(i)$ Zeiger auf den Vorgängerzustand von Zustand s_i zum Zeitpunkt t, Seite 15

$\delta_i(j,k)$ Lokale Pfadwahrscheinlichkeit für Zeitpunkt k an Zustand s_i und mit Vorgängerzustand s_j, Seite 40

Sonstige Definitionen

i Indexvariable für die $i \in \mathbb{N}$ gilt, Seite 13

j Indexvariable für die $j \in \mathbb{N}$ gilt, Seite 13

k Eine beliebige Konstante. Es gilt $k \in \mathbb{N}$, Seite 13

l Längen-/Größenvariable für die $l \in \mathbb{N}_0^+$ gilt, Seite 13

$P(A)$ Wahrscheinlichkeit für ein Ereignis A, Seite 13

$P(A|B)$ Wahrscheinlichkeit für ein Ereignis A unter der Voraussetzung, dass Ereignis B bereits eingetreten ist, Seite 13

$A \models B$ Die Beschreibung B wird durch A modelliert, Seite 15

ε Es gilt, dass $\varepsilon > 0$ aber beliebig klein ist, Seite 19

GHMM Generalized Hidden Markov Modell, Seite 20

GPHMM Generalized Pair Hidden Markov Modell, Seite 20

$(a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n)$ Ein Vektor, der als Pfad interpretiert wird, Seite 39

$\vec{a}$ Ein Vektor mit der Bezeichnung a. Man schreibt auch $a = (a_1, a_2, \ldots, a_n)$ wobei n die Dimension des Vektors ist, Seite 39

M Eine beliebige Menge, Seite 39

M^* Der Kleensche Abschluss einer Menge M, Seite 39

l_{align} Länge eines Alignments, Seite 52

l_{max} Länge der längsten Eingabesequenz, Seite 52

ϵ Kante im Graph G, Seite 78

$\mathbb{E}$ Kantenmenge, Seite 78

$\mathbb{V}$ Knotenmenge, Seite 78

G Gerichteter Graph, Seite 78

v, v' Knoten im Graph G, Seite 78

ω Funktionssymbol, Seite 8

m Längen-/Größenvariable für die meist $m \in \mathbb{N}_0^+$ gilt, Seite 8

n Längen-/Größenvariable für die meist $n \in \mathbb{N}_0^+$ gilt, Seite 8

X Zufalls-, Sequenz- oder Eingabevariable $X = (x_1, \ldots, x_l)$., Seite 8

x_i Elementarereignisse einer Zufallsvariable. Meistens gilt $x_i \in \{0, 1\}$ oder $x_i \in \Sigma$, Seite 8

Y Zufalls-, Sequenz- oder Eingabevariable $Y = (y_1, \ldots, y_l)$, Seite 8

y_i Elementarereignisse einer Zufallsvariable bzw. Teile einer Eingabe. Meistens gilt $y_i \in \{0, 1\}$ oder $y_i \in \Sigma$, Seite 8

l_ϵ Die Länge einer Kante ϵ im Graph G, Seite 80

$l_{minDist}$ Mindestlänge einer Kante ϵ im Graph G, Seite 81

d_ϵ Verweildauerwahrscheinlichkeit der Kante ϵ im Parsegraphen, Seite 82

e_ϵ Emissionswahrscheinlichkeit der Kante ϵ im Parsegraphen, Seite 82

$p(\epsilon)$ Kantenwahrscheinlichkeit im Parsegraphen, Seite 82

LWP_i Lokale Pfadwahrscheinlichkeit für ein Signal i, Seite 86

t_ϵ Transitionswahrscheinlichkeit der Kante ϵ im GPHMM, Seite 86

D Beliebige Matrix, Seite 9

$d_{i,j}$ Elemente der Matrix D, Seite 9

1 Einleitung

Mit den Projekten zur Sequenzierung des menschlichen Genoms und deren Abschluss im Jahr 2003 ist die Bioinformatik verstärkt in den Fokus der Öffentlichkeit gerückt und ist dadurch heute eines der bekannteren Teilgebiete der Informatik. Unabhängig von solch spektakulären Projekten ist die Informatik, wie bei eigentlich allen Naturwissenschaften, aus der modernen Biologie und hier vor allem aus der bereits genannten Genom- aber auch Proteomforschung seit längerem nicht mehr weg zu denken. Gleichzeitig bietet sie als relativ junge Disziplin ein weites und ergiebiges Spektrum an Möglichkeiten der Forschung. Im Rahmen der Projektgruppe 487 - „Methoden der Computational Intelligence in der Bioinformatik“ - sollen für ausgewählte Probleme der Biologie bzw. Bioinformatik algorithmische Lösungen aus dem Bereich der Evolutionären Algorithmen entwickelt und implementiert, sowie diese mit gegebenenfalls vorhandenen Lösungen verglichen werden. Eine Projektgruppe ist eine einjährige praktische Arbeit, die von zwölf Studenten zu einem übergeordneten Thema durchgeführt wird. Der vorliegende Endbericht gibt einen umfassenden Einblick über die von den PG-Teilnehmern erworbenen Grundkenntnisse, sowie deren Anwendung und Analyse während des zweisemestrigen PG-Verlaufs. Dabei gliedert sich der Bericht in die drei Kapitel Grundlagen, Anwendungsgebiete und Zusammenfassung.

Im Grundlagenkapitel werden zunächst die notwendigen Grundkenntnisse beschrieben, die zum Verständnis der praktischen Themen nötig sind. Dazu gehören die Grundbegriffe der Molekularbiologie, wie DNA oder Proteine, die Grundlagen der Bioinformatik mit dessen spezifischen Problemstellungen und eine Methode aus dem Bereich der Computational Intelligence, die Evolutionären Algorithmen, die bei allen im zweiten Kapitel beschriebenen Verfahren angewendet werden. Zudem werden die theoretischen Grundlagen für Hidden Markov Modelle, die zur Betrachtung stochastischer Prozesse verwendet werden, eingeführt.

Das zweite Kapitel widmet sich dem praktischen Teil der Projektarbeit. Hierzu wurde das breite Spektrum der Probleme der Bioinformatik in mehreren Teilgruppen bearbeitet. Es wird dargestellt, wie mit Hilfe von Profile HMM Sequenzen aligniert werden, wie eine Genvorhersage durchgeführt wird, wie mit Hilfe von HMM-Mustern Proteinstrukturen gefunden werden oder wie phylogentische Bäume erstellt werden können. Die einzelnen Gruppen stellen ihre Problemstellung vor, beschreiben die Herangehensweise und präsentieren die erzielten Ergebnisse.

Abschließend findet eine Diskussion der erarbeiteten Ergebnisse in einer Zusammenfassung statt. Es werden die gesteckten Ziele innerhalb der Gruppenarbeit mit den erreichten Zielen verglichen und die erarbeiteten Kenntnisse sowie die erworbenen Fähigkeiten mit den Zielen einer Projektgruppe in der Diplomprüfungsordnung formulierten Zielen verglichen und bewertet.

2 Grundlagen

2.1 Molekularbiologie

Eine Zelle ist die kleinste selbstständig lebensfähige Funktionseinheit aller Lebewesen (Elementarorganismus). Sie ist ein sehr komplexes System aus vielen, sich frei bewegenden Untereinheiten. Sie enthält nicht nur alle Informationen, um sich selbst zu duplizieren, sondern auch alle nötigen Komponenten, um die Rohstoffe, die dafür nötig sind, zu sammeln und zu verarbeiten.

In allen Lebensformen finden sich DNA, RNA und Proteine [Löf01]. Die DNA enthält die Erbinformationen und speichert damit den Aufbau aller körpereigenen Proteine. Sie wurde 1896 von Johann Friedrich Miescher entdeckt, als er sie aus dem Zellkern weißer Blutkörperchen extrahierte. Anfang des 20. Jahrhunderts war zwar bekannt, aus welchen Basen sich die DNA zusammensetzte (Adenin, Thymin, Guanin und Cytosin), allerdings wurde lange Zeit das kodierende Muster der DNA nicht erkannt und die Folge von Basen für eine periodische Abfolge gehalten. Die fünfte entdeckte Base war Uracil, die Thymin stark ähnelt. Die DNA enthält dort Thymin, wo die RNA Uracil enthält. Das Interesse an der DNA erhöhte sich schlagartig, als Oswald Avery 1944 mit seinen Kollegen nachwies, dass die Gene tatsächlich in der DNA gespeichert und bestimmte Teilabschnitte der DNA sind.

Damals war jedoch noch nicht bekannt, wie die Informationen aus der DNA, die sich im Zellkern befindet, an den Ort der Proteinsynthese innerhalb der Zelle gelangt. Aus diesem Grund vermutete man eine noch unbekannte Substanz, die als Träger die genetischen Informationen vom Zellkern zu den Ribosomen transportieren konnte, wo die Proteine synthetisiert werden. Die Vermutung, dass RNA ein Zwischenprodukt bei der Synthetisierung der Proteine aus der DNA ist, wurde dadurch gefestigt, dass in den Ribosomen RNA enthalten ist.

Im Jahr 1960 fanden Jerard Hurwitz und Samuel Weiss ein Protein, welches DNA als Schablone benutzt und den entsprechenden Messenger-RNA (mRNA)-Strang erstellt. Dieser Prozess wird Transkription und das zugehörige Protein wird RNA-Polymerase genannt. Die Transkription der Gene in mRNA ist ein von dem Zustand der Zelle abhängiger Prozess, bei dem nicht alle Gene zu jeder Zeit die entsprechenden Proteine erzeugen [Löf01]. Dieser Prozess, bei dem unterschiedliche Gene aktiviert werden und sich so über die Erzeugung der Proteine bestimmte Merkmale bilden, wird Genexpression genannt. Aus einer mRNA entstehen in einem weiteren Schritt, der Spleißen genannt wird, mehrere unterschiedliche, neu zusammengeschnittene mRNA-Stücke. Hierbei werden Bereiche, die Proteine kodieren (Exons), aneinandergefügt und nicht kodierende Bereiche (Introns) herausgeschnitten. Die so entstandene mRNA wird in weiteren Schritten als Bauplan für Proteine genutzt.

In den späten 60er Jahren wurde die sogenannte Triple-Regel, die besagt, dass drei aufeinander folgende Basen der DNA eine Aminosäure kodieren, bestätigt [Löf01]. Damit war das Problem gelöst, wie aus dem aus vier Buchstaben bestehenden Alphabet der DNA bzw. RNA ein eine Sequenz über einem Alphabet von zwanzig Buchstaben erzeugt werden kann. Dieses Kodierungsschema ist in Abbildung 2.1 in Form der sogenannten „Kodonsonne“ dargestellt.

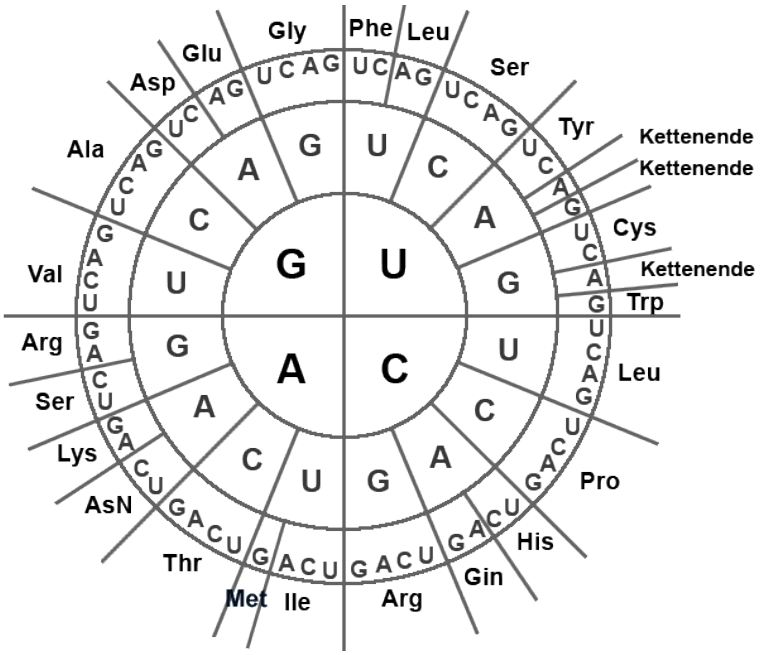

Abbildung 2.1: Kodonsonne zum Ablesen der entsprechenden Aminosäure bei einer gegebenen mRNA-Sequenz. AUG ist hierbei sowohl das Startkodon für ein Protein als auch die Kodierung für die Aminosäure Methionin. UAA, UAG und UGA signalisieren das Ende des Proteins.

Proteine erfüllen in der Zelle ein breites Spektrum an Aufgaben. Die Funktion des Proteins wird durch seine räumliche Struktur bestimmt [Löf01]. Die Aminosäuresequenz des Proteins wird als Primärstruktur bezeichnet. Dessen Bestimmung wird Sequenzierung genannt. Darüber hinaus gibt es weitere Klassifikationen der räumlichen Struktur von Proteinen:

Sekundärstruktur: Durch die Bindungen zwischen den einzelnen Aminosäuren kommen in vielen Proteinen wiederkehrende Strukturmotive vor. Häufige Formen sind hierbei eine rechtsgewundene Schraube, die sich α-Helix nennt oder das β-Faltblatt, bei dem sich die Aminosäuresequenz abwechselnd in unterschiedliche Richtungen faltet.

Tertiärstruktur: Die Tertiärstruktur bildet die nächst höhere Organisationsstufe und beschreibt die Ausbildung von Kombinationen aus den Klassen der Sekundärstrukturen. Durch die Kombination von α-Helices und β-Faltblättern den dazwischen

liegenden Schleifen und anderen Strukturelementen bildet sie eine stabile Raumstruktur.

Quartärstruktur: Bei der Quartärstruktur kommen Proteinketten mit jeweils eigener Primär-, Sekundär- und Tertiärstruktur zu einer funktionalen Einheit zusammen (z.B. Globulin als Teil des Hämoglobin).

Die elementaren Bausteine der Proteine sind die Aminosäuren, die durch Peptidbindung (s. hierzu [Löf01]) verbunden werden. Aminosäuren können bzgl. ihrer chemischen Eigenschaften in die Klassen hydrophob (neutral) und polar (geladen) eingeteilt werden. Für ihre unterschiedlichen chemischen Eigenschaften sind die Seitenketten verantwortlich. Es ergibt sich folgende Einteilung [Löf01]:

hydrophob: Glycin, Alanin, Valin, Leucin, Isoleucin, Methionin, Prolin, Phenylalanin und Tryptophan

polar: Serin, Threonin, Asparagin, Glutamin, Cystein, Selenocystein, Tyrosin, Aspartat, Glutamat, Histidin, Lysin und Arginin

2.2 Bioinformatik

Die Bioinformatik ist eine Wissenschaft, die sich mit informatischen Grundlagen und Anwendungen der Speicherung, Organisation und Analyse von biologischen Daten befasst. Sie ist ein interdisziplinäres Feld bestehend aus Biologie, Informatik, Mathematik und Statistik, das sich mit der Analyse biologischer Sequenzen und der Vorhersage von Struktur und Funktion von biologischen Makromolekülen beschäftigt. Mittlerweile sind jedoch viele Aufgaben in der biologischen und medizinischen Forschung hinzugekommen. Ein zentrales Thema der Bioinformatik ist die Sequenzanalyse. Drei wichtige Verfahren werden im Folgenden erläutert.

Sequenzalignierung: Genetische Informationen sind in der Nukleotidsequenz der DNA kodiert. Spezialisierungen und parallele Entwicklungen erfolgen durch Veränderungen entsprechender Nukleotidfolgen in der DNA. Bei der Untersuchung einer DNA-Sequenz kann die evolutionäre Entwicklung in Form von phylogenetischen Bäumen nachvollzogen werden (s. Lesk [Les03]). Dabei handelt es sich um Graphen, die geschichtliche Verwandschaftsbeziehungen verschiedener Spezies modellieren. Des Weiteren besitzen homologe, also gleiche Abschnitte der Proteine häufig identische Sekundärstrukturen. Somit lassen sich Hypothesen über die Struktur und Funktion eines unbekannten Proteins unter Verwendung bereits bekannter Proteine aufstellen. Die in Kapitel 2.3 beschriebenen Methoden werden in diesen beiden Bereichen eingesetzt.

Genvorhersage: Die Genvorhersage befasst sich mit dem Aufspüren der für ein Protein kodierenden Regionen (vgl. Pachter [Pac02]). Hierbei müssen kodierende Regionen von den nicht kodierenden unterschieden werden. Algorithmen zur Genvorhersage

werden in Ab initio-Verfahren und homologiebasierte Verfahren eingeteilt. Während bei den Ab initio-Verfahren nach Signalen in der Gensequenz gesucht wird, basiert der Homologieansatz auf dem Vergleich schon bekannter Sequenzen.

Strukturvorhersage: Die Form und Funktion eines Proteins wird durch seine Sequenz bestimmt. Wie bei der Genvorhersage werden zwei algorithmische Ansätze zur Vorhersage von Proteinstrukturen unterschieden: der Homologieansatz und die Ab initio-Methode.

2.3 Sequenzalignierung

Ziel einer Sequenzalignierung ist es, Sequenzen durch Einfügung von Leerzeichen, so genannten Gap-Symbolen oder Lücken, auf eine einheitliche Länge zu bringen. Dabei sollen möglichst viele gleichartige Symbole an den gleichen Positionen der alignierten Sequenzen stehen. Anhand eines Alignments lassen sich Gemeinsamkeiten in den Sequenzen erkennen und bewerten. Befinden sich an gleicher Position des Alignments gleiche Zeichen, so ist dies ein Match, andernfalls liegt ein Mismatch vor. Ein Alignment kann dabei zu je zwei Sequenzen (Paarweises Alignment) oder zu mehreren Sequenzen (Multiples Alignment) bestimmt werden.

2.3.1 Alignmentvarianten

Man unterscheidet die folgenden Alignmentvarianten nach deren Funktion:

Globales Alignment: Beim Globalen Alignment werden die Sequenzen über ihre gesamte Länge verglichen. Es findet damit hauptsächlich Anwendung bei eng verwandten, also sehr ähnlichen Sequenzen.

Lokales Alignment: Beim Lokalen Alignment werden Bereiche maximaler Ähnlichkeit in den Sequenzen gesucht. Hierbei können auch stark unterschiedliche Sequenzen untersucht werden.

Semiglobales Alignment: Für Sequenzen mit deutlichen Längenunterschieden werden Semiglobale Alignments erstellt. Das Alignment wird, wie beim Globalen Alignment, über die gesamte Sequenzlänge bewertet, jedoch gehen Lücken am Anfang und am Ende der kürzeren Sequenz nicht in die Bewertung mit ein.

2.3.2 Bewertung von Alignments

Es gibt mehrere Möglichkeiten, die Qualität von Alignments zu bestimmen. Zwei gängige Verfahren sind:

2.3.2.1 Uniforme Bewertung

Es gibt feste Bewertungen für Match, Mismatch und Lücken. Für jede Spalte wird die Bewertung bestimmt und aufsummiert. Die Summe der Spaltenbewertung ergibt so die Gesamtbewertung des Alignments.

2.3.2.2 Bewertung mit Substitutionsmatrizen

Im Gegensatz zur uniformen Bewertung geben Substitutionsmatrizen jeder Symbolkombination eine eigenständige Bewertung. Der Wert in jedem Matrixelement spiegelt die biologische Ähnlichkeit der zugehörigen Symbole wieder. Dabei wird zwischen Substitutionsmatrizen für die verschiedenen Alignmentvarianten unterschieden. Bei Aminosäuren sind dies beispielsweise die PAM-Substitutionsmatrix (Percent Accepted Mutations) für Globale und die BLOSUM-Substitutionsmatrix (BLOcks SUbstitution Matrix) für Lokale Alignments.

2.3.3 Die Methode der Dynamischen Programmierung

Bevor in den Abschnitten 2.3.4 und 2.3.5 dieses Kapitels optimale Lösungsstrategien für Lokale und Globale Alignments vorgestellt werden, soll hier zunächst der ihnen zugrundeliegende Ansatz der Dynamischen Programmierung (DP) erläutert werden.

Die DP ist ein durch den amerikanischen Mathematiker Richard Bellman [Bel57] geprägtes Paradigma zum algorithmischen Lösen von Optimierungsproblemen. "Dynamischßteht dabei für sequentiell und PProgrammierungffür das eigentliche Optimieren. DP ist dann auf ein Optimierungsproblem anwendbar, wenn dessen Lösung sich aus optimalen Lösungen kleinerer, nicht notwendigerweise unabhängiger, Teilprobleme zusammensetzt. Dieses Konzept wird Bellmansches Optimalitätsprinzip genannt. Der Ablauf eines DP-Algorithmus besteht üblicherweise aus folgenden Schritten:

1. Charakterisieren der optimalen Lösung
2. Rekursives Definieren des Wertes einer optimalen Lösung
3. Berechnung des optimalen Wertes
4. Finden der optimalen Lösung durch während der Berechnung gespeicherte Informationen

Die zentrale Aufgabe besteht also darin zu beschreiben, in welcher Art und Weise Teillösungen zusammengesetzt werden, um die Lösung für ein größeres Problem zu bilden. Dieser Zusammenhang wird durch die Bellmansche Optimalitätsgleichung ausgedrückt. Die Informationen, die zur Lösung der einzelnen Teilprobleme nötig sind, werden während der Berechnung schrittweise ergänzt. Um mehrfache Berechnungen zu verhindern, werden die Lösungen der Teilprobleme in einer Matrix gespeichert.

2.3.4 Der Needleman-Wunsch-Algorithmus

Ein Algorithmus zur Berechnung eines (optimalen) Globalen Alignments wurde 1970 von Saul Needleman und Christian Wunsch [Nee70] veröffentlicht (siehe Kapitel 2.3.1). Im Folgenden wird die Grundversion des Needleman-Wunsch-Algorithmus (NW) für Paarweise Alignments vorgestellt. Die Eingabesequenzen seien $X = x_1 \dots x_m$ und $Y = y_1 \dots y_n$, mit x_i, y_j aus einem gegebenen Ausgabealphabet Σ.

2.3.4.1 Grundfunktion des Algorithmus

Der zugrunde liegende Algorithmus besteht aus drei Teilen. Zuerst wird die Matrix initialisiert, danach werden die Matrixelemente berechnet. Zuletzt wird das Alignment über eine Rückverfolgung des Berechnungsweges aufgebaut.

2.3.4.2 Initialisierung der Matrix

Die Dimension der Matrix $D = (d_{i,j})_{0 \leq i \leq n, 0 \leq j \leq m}$ ist abhängig von den Längen der Eingabesequenzen. Die Spalten werden den jeweiligen Symbolen der Sequenz X, die Zeilen den Symbolen der Sequenz Y zugeordnet. Die Matrix wird wie folgt initialisiert:

$$d_{i,0} = -i \quad \text{mit } i = 0, \dots n$$
$$d_{0,j} = -j \quad \text{mit } j = 0, \dots m.$$

Abbildung 2.2 verdeutlicht den Aufbau.

		x_1	x_2	x_3	x_4	...	x_m
	0	-1	-2	-3	-4	...	-m
y_1	-1						
y_2	-2						
y_3	-3						
y_4	-4						
...	...						
y_n	-n						

Abbildung 2.2: Leere Matrix für den Needleman-Wunsch-Algorithmus. Die jeweils erste Zeile und Spalte wird zur Initialisierung mit von 0 absteigenden, ganzen Zahlen gefüllt.

2.3.4.3 Berechnung der Matrixelemente

Die Bewertung von Alignments erfolgt durch eine Funktion $\omega : \Sigma' \times \Sigma' \to \mathbb{R}$, wobei $\Sigma' = \Sigma \cup \{-\}$ gilt. Hierbei bezeichnet „$-$“ das Gap-Symbol.
Folgende vier Fälle sind zu unterscheiden:

- Einfügen einer Lücke in Y: $\omega(x_i, -)$
- Einfügen einer Lücke in X: $\omega(-, y_j)$

- Match von Elementen: $\omega(x_i, y_j)$ mit $x_i = y_j$
- Mismatch/Replace von Elementen: $\omega(x_i, y_j)$ mit $x_i \neq y_j$

Einfügungen von Lücken und Mismatches sollten in der Bewertung bestraft, Matches belohnt werden. Beispielbewertungen wären -2 für das Einfügen einer Lücke, -1 für ein Mismatch und $+1$ für ein Match von zwei Symbolen. Die Einträge $d_{i,j}$ der Matrix D lassen sich folgendermaßen berechnen:

$$d_{i,j} = max \begin{cases} d_{i-1,j} & + \ \omega(x_i, -) & \text{(a)} \\ d_{i,j-1} & + \ \omega(-, y_j) & \text{(b)} \\ d_{i-1,j-1} & + \ \omega(x_i, y_j) & \text{(c).} \end{cases} \tag{2.1}$$

Die Berechnung beginnt beim Matrixelement $(1, 1)$ und geschieht danach zeilen- oder spaltenweise. Der für den Wert von $d_{i,j}$ ausschlaggebene Eintrag wird als Vorgänger von $d_{i,j}$ bezeichnet. Die mit (a)-(c) bezeichneten Fälle entsprechen dabei:

- (a) Lücke in Sequenz Y einfügen
- (b) Lücke in Sequenz X einfügen
- (c) Match oder Mismatch von x_i und y_j

Eine ausgefüllte Beispielmatrix für die Sequenzen X = ACGCAA und Y = ATAGCA findet sich in Abbildung 2.3.

	-	A	T	A	G	C	A
-	0	-1	-2	-3	-4	-5	-6
A	-1	1	0	-1	-2	-3	-4
C	-2	0	0	-1	-2	-1	-2
G	-3	-1	-1	-1	0	-1	-2
C	-4	-2	-2	-2	-1	1	0
A	-5	-3	-3	-1	-2	0	2
A	-6	-4	-4	-2	-2	-1	1

Abbildung 2.3: Mit Beispielzahlen gefüllte Matrix des Needleman-Wunsch-Algorithmus. Ein Match ist in dem Beispiel mit einem Punkten bewertet. Für eine Lücke werden zwei Punkte abgezogen und ein Mismatch zieht einen Punkt ab.

2.3.4.4 Rückverfolgung

Das eigentliche Alignment wird während der Rückverfolgung erstellt. Der Berechnungsweg für das Matrixelement $d_{n,m}$ wird bis Position $(0, 0)$ zurückverfolgt. Dieser Weg kann entweder schon während des Aufbaus gespeichert oder nachträglich in linearer Laufzeit errechnet werden. Der Aufbau des Alignments geschieht hierbei durch Betrachtung der Position des jeweiligen Vorgängers:

- Schräg nach links oben: Match oder Replace
- Nach links: Einfügen einer Lücke in Sequenz X
- Nach oben: Einfügen einer Lücke in Sequenz Y

Als Beispiel findet sich in Abbildung 2.4 die Matrix aus dem vorherigen Abschnitt für die Sequenzen X = ACGCAA und Y = ATAGCA. Das resultierende Alignment ist hier dargestellt:

```
A - C G C A A
A T A G C A -
```

	-	A	T	A	G	C	A
-	0	-1	-2	-3	-4	-5	-6
A	-1	1	0	-1	-2	-3	-4
C	-2	0	0	-1	-2	-1	-2
G	-3	-1	-1	-1	0	-1	-2
C	-4	-2	-2	-2	-1	1	0
A	-5	-3	-3	-1	-2	0	2
A	-6	-4	-4	-2	-2	-1	1

Abbildung 2.4: Gefüllte Beispielmatrix des Needleman-Wunsch-Algorithmus. Der gelb markierte Pfad ist der Weg zur höchsten Zahl in der Matrix. Dieser Weg wird zur Bestimmung des Alignments benötigt.

2.3.4.5 Laufzeitanalyse

Die Laufzeit des Algorithmus ist $O(n \cdot m) = O(k^2)$, wobei $k{=}n{+}m$ die Länge der Eingabe ist. Die Laufzeit wächst also quadratisch in der Eingabelänge. Der Speicherplatz, der benötigt wird, welcher sich aus der Größe der Matrix und der Größe des maximalen Matrixelements ergibt, ist somit $O(n \cdot m \cdot \log(n+m))$.

2.3.5 Der Smith-Waterman-Algorithmus

Der Smith-Waterman-Algorithmus (SW), veröffentlich von Smith und Waterman [SW81], ist ein Verfahren zur Berechnung von Lokalen Alignments, welches ebenfalls auf dynamischer Programmierung basiert. Er stellt eine Erweiterung des NW-Algorithmus dar. Hier werden jedoch während der Berechnung keine Strafen für große Lücken vergeben und es wird versucht, die Sequenzen so zu alignieren, dass Ähnlichkeiten maximiert und nicht Unterschiede minimiert werden.

2.3.5.1 Grundfunktion des Algorithmus

Prinzipiell arbeitet der SW-Algorithmus sehr ähnlich zum NW-Algorithmus. Im Weiteren werden deshalb die Unterschiede zu diesem beschrieben.

2.3.5.2 Erstellen der leeren Matrix

Die Initialisierung unterscheidet sich von der des NW-Algorithmus insofern, dass die jeweils erste Spalte und Zeile komplett mit Nullen gefüllt wird (siehe Abbildung 2.5).

		x_1	x_2	x_3	x_4	...	x_m
	0	0	0	0	0	...	0
y_1	0						
y_2	0						
y_3	0						
y_4	0						
...	...						
y_n	0						

Abbildung 2.5: Leere Matrix für den Smith-Waterman-Algorithmus. Die jeweils erste Zeile und Spalte sind zur Initialisierung mit Nullen gefüllt.

2.3.5.3 Berechnung der Matrixelemente

Die Bewertungsfunktion wird gegenüber 2.1 wie folgt erweitert:

$$d_{i,j} = max \begin{cases} 0 \\ d_{i-1,j} + \omega(x_i, -) \\ d_{i,j-1} + \omega(-, y_j) \\ d_{i-1,j-1} + \omega(x_i, y_j). \end{cases} . \tag{2.2}$$

Dadurch können im Unterschied zum NW-Algorithmus keine negativen Matrixeinträge entstehen. Hierdurch wird verhindert, dass Teilabschnitte der Sequenz schlechter bewertet werden, wenn ihnen viele Leerzeichen oder Mismatches vorausgehen. Eine ausgefüllte Beispielmatrix für die Sequenzen X = ACGCAA und Y = ATAGCA findet sich in Abbildung 2.6.

2.3.5.4 Rückverfolgung

Der Unterschied bei der Rückverfolgung liegt darin, dass der Startpunkt des Rückweges nicht bei Element $d_{n,m}$ beginnt, sondern beim Matrixelement mit maximalem Wert. Der Pfad wird des Weiteren nur solange zurück verfolgt, bis eine Eintrag mit Wert Null erreicht wird. Als Beispiel findet sich in Abbildung 2.7 die Matrix aus dem vorherigen Abschnitt für die Sequenzen X = ACGCAA und Y = ATAGCA.

Das entsprechende Alignment ist hier dargestellt:

```
-  -  A  C  G  C  A  A
A  T  A  -  G  C  A  -
```

	-	A	T	A	G	C	A
-	0	0	0	0	0	0	0
A	0	3	2	3	2	1	3
C	0	2	2	2	2	5	4
G	0	1	1	1	5	4	4
C	0	0	0	0	4	8	7
A	0	3	2	3	3	7	11
A	0	3	2	5	4	6	10

Abbildung 2.6: Mit Beispielzahlen gefüllte Matrix des Smith-Waterman-Algorithmus. Ein Match ist in dem Beispiel mit drei Punkten bewertet. Die Strafe für eine Lücke oder ein Mismatch ist ein Punkt.

	-	A	T	A	G	C	A
-	0	0	0	0	0	0	0
A	0	3	2	3	2	1	3
C	0	2	2	2	2	5	4
G	0	1	1	1	5	4	4
C	0	0	0	0	4	8	7
A	0	3	2	3	3	7	11
A	0	3	2	5	4	6	10

Abbildung 2.7: Gefüllte Beispielmatrix des Smith-Waterman-Algorithmus. Der gelb markierte Pfad ist der Weg zur höchsten Zahl in der Matrix. Dieser Weg wird zur Bestimmung des Alignments benötigt.

2.3.5.5 Laufzeitanalyse und Speicherplatzbedarf

Laufzeit und Speicherplatzbedarf entsprechen denen des NW-Algorithmus (siehe Kapitel 2.3.4.5).

2.4 Hidden Markov Modelle

Ein Hidden Markov Modell (HMM) ist ein stochastisches Modell, das sich durch zwei Zufallsprozesse beschreiben lässt. Der erste Zufallsprozess entspricht dabei einer Markov-Kette, die durch Zustände und Übergangswahrscheinlichkeiten gekennzeichnet ist. Die Zustände der Kette sind von außen jedoch verborgen. Ein zweiter Zufallsprozess erzeugt die zu jedem Zeitpunkt beobachteten Ausgangssymbole gemäß einer zustandsabhängigen

Wahrscheinlichkeitsverteilung. Formal lässt sich ein Hidden Markov Modell durch ein Fünf-Tupel

$$HMM = (S, \Sigma, T, \Pi, E)$$

beschreiben. Dabei gilt:

Zustandsmenge: $S = \{s_1, \ldots, s_n\}$.

Emissionsalphabet: $\Sigma = \{\sigma_1, \ldots, \sigma_l\}$.

Transitionsmatrix: T mit $t_{i,j} \in T$ und $1 \leq i, j \leq n$.

Startverteilung: $\Pi = (\pi_1, \ldots, \pi_n)$.

Emissionsmatrix: E mit $e_{i,j} \in E$ und $i \in \{1, \cdots, n\}$ sowie $j \in \{1, \cdots, l\}$.

Die Transitionswahrscheinlichkeiten geben an, mit welcher Wahrscheinlichkeit von einem Zustand in einen anderen übergegangen wird. Die Emissionswahrscheinlichkeiten geben an, mit welcher Wahrscheinlichkeit ein bestimmtes Symbol ausgegeben wird. Ein einführendes Tutorial zu HMM von Rabiner ist [Law89]. Für dieses Modell werden die folgenden drei Problemklassen definiert:

Evaluierungsproblem: Ein Evaluierungsalgorithmus ermittelt für eine gegebene Sequenz und ein bestehendes HMM die maximale Wahrscheinlichkeit, mit der das HMM diese Sequenz emittiert. Ein Algorithmus für dieses Problem ist der Forward-Algorithmus (siehe Kapitel 2.4.2.4).

Trainingsproblem: Für eine Sequenz von Emissionen soll ein HMM so trainiert werden, dass die Sequenz durch das trainierte HMM mit höherer Wahrscheinlichkeit emittiert wird. Hierzu werden die Übergangswahrscheinlichkeiten und Emissionswahrscheinlichkeiten verändert. Der Baum-Welch-Algorithmus (siehe Kapitel 2.4.2) ist ein Ansatz zur Lösung dieses Problems.

Dekodierungsproblem: Für eine gegebene Sequenz von sichtbaren Zuständen wird die wahrscheinlichste Folge von verborgenen Zuständen im HMM ermittelt. Dieses Problem wird mit Hilfe des Viterbi-Algorithmus gelöst, welcher im Folgenden (siehe Kapitel 2.4.1) näher betrachtet wird.

Für eine Markov-Kette (Andrei A. Markov, [Mar06]), auch „Markov-Prozess“ genannt, mit $S = (s_1, s_2, \cdots, s_n)$ als Zustandsmenge gilt, dass die Wahrscheinlichkeit, sich in einem Zustand s_k zu befinden, nur vom Vorgängerzustand abhängt:

$$P(s_k|s_{k-1}, \cdots, s_1) = P(s_k|s_{k-1}).$$

Diese Eigenschaft, die Markov-Eigenschaft, ist Grundlage vieler Algorithmen, die auf Hidden Markov Modellen arbeiten.

2.4.1 Der Viterbi-Algorithmus

Der Viterbi-Algorithmus wurde 1967 von Andrea Viterbi [Vit67] entwickelt und löst das Dekodierungsproblem durch Dynamische Programmierung (siehe Kapitel 2.3.3).

2.4.1.1 Trellis

Im Folgenden soll der Ablauf des Viterbi-Algorithmus vorgestellt werden. Da ein HMM keine Informationen über den zeitlichen Verlauf enthält, lässt sich nicht feststellen, wie oft ein Zustand bei einem Durchlauf besucht wurde. Aus diesem Grund wird eine andere Betrachtungsweise des HMM vorgestellt, die eine Verfolgung der zeitlichen Abläufe ermöglicht. Dieser neue Graph wird Trellis (dt. Spalier) genannt. Dabei wird die Zustandsmenge für jeden Zeitpunkt kopiert und spaltenweise nebeneinander geschrieben. Die Transitionen verlaufen nicht mehr zu den Zuständen innerhalb einer Spalte, sondern zeigen auf die äquivalenten Zustände der nächsten Spalte. Eine Transition ist in diesem Fall ein Übergang von einer Spalte in die nächste, wodurch eine zeitliche Abfolge modelliert wird. Abbildung 2.8 veranschaulicht die Konstruktion eines solchen Trellis.

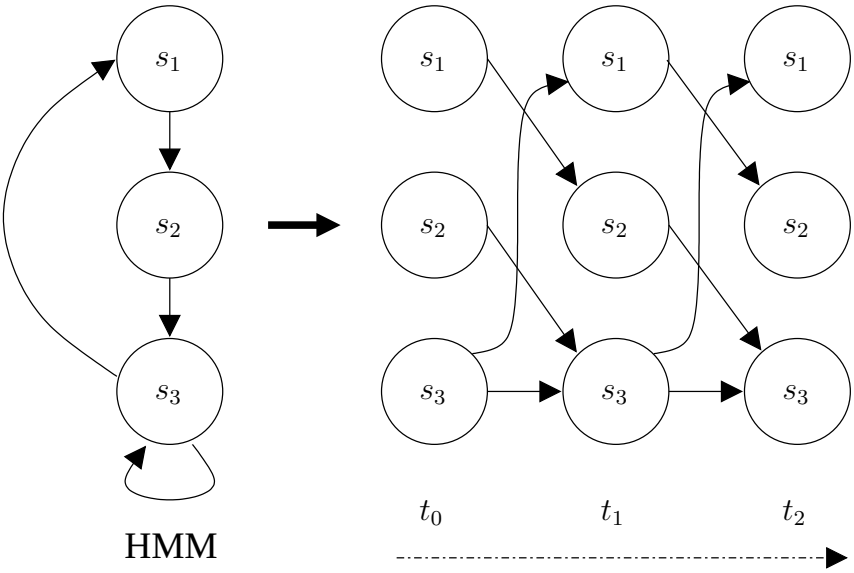

Abbildung 2.8: Die Zustandsmenge wird im Trellis spaltenweise kopiert und die Transitionen werden über die Spalten hinweg verbunden.

2.4.1.2 Datenstrukturen

Der Viterbi-Algorithmus arbeitet mit zwei Datenstrukturen. Zum einen wird für jeden Zustand im Trellis ein Wert abgespeichert, der die Wahrscheinlichkeit angibt, zu einem Zeitpunkt k unter der gegebenen Emissionssequenz im Zustand s_i zu sein. Dieser Wert δ ist die lokale Pfadwahrscheinlichkeit. Zum anderen wird für jeden Zustand ein Zeiger

ϕ abgelegt, der angibt, aus welchem Vorgängerzustand sich der Wert berechnet hat.

$\delta_k(i) \models$ lokale Pfadwahrscheinlichkeit für Zeitpunkt k an Zustand s_i.
$\phi_k(i) \models$ Zeiger auf den Vorgängerzustand von Zustand s_i zum Zeitpunkt k.

2.4.1.3 Initialisierung

Bei der Initialisierung werden die lokalen Pfadwahrscheinlichkeiten für den ersten Zeitpunkt berechnet. Diese Startwerte ergeben sich aus der Startverteilung Π und den Emissionswahrscheinlichkeiten. Für eine Eingabesequenz $X = (x_1, \ldots, x_l)$ berechnet sich δ folgendermaßen:

$$\delta_1(i) = e_{s_i,x_1} \cdot \pi_i.$$

Für die Zustände der ersten Iteration berechnet sich die lokale Pfadwahrscheinlichkeit als Produkt der Emissionswahrscheinlichkeit e_{s_i,x_1} und der Startwahrscheinlichkeit des Zustands.

2.4.1.4 Lokale Pfadberechnung

In jeder Iteration k wird für jeden Zustand die lokale Pfadwahrscheinlichkeit berechnet. Dabei gilt:

$$\delta_k(i) = \max_j(\delta_{k-1}(j) \cdot t_{j,i} \cdot e_{s_i,x_k}).$$

Die neue lokale Pfadwahrscheinlichkeit zum Zeitpunkt k berechnet sich aus dem Produkt der Wahrscheinlichkeit des Zustandes der letzten Iteration $k-1$, der Transitionswahrscheinlichkeit und der Emissionswahrscheinlichkeit für diesen Zustand mit dem emittierten Symbol. Daraus wird das Maximum gesucht und als neuer Wert eingetragen. Gibt es keinen Übergang zwischen den Zuständen s_i und s_j, so wird die Übergangswahrscheinlichkeit auf 0 gesetzt. Außerdem wird für jeden Zustand in einer zweiten Datenstruktur gespeichert, welcher Vorgängerzustand s_j der maximale war. Es gilt:

$$\phi_k(i) = \operatorname*{argmax}_j(\delta_{k-1}(j) \cdot t_{j,i}).$$

2.4.1.5 Rückverfolgung

Wurden für alle Zeitpunkte die lokalen Pfadwahrscheinlichkeiten ermittelt, wird der Zustand als maximaler Zustand gewählt, der in der letzten Iteration l die höchste Wahrscheinlichkeit besitzt. Es gilt:

$$i_l = \operatorname*{argmax}_j(\delta_l(j)).$$

Von diesem Zustand ausgehend wird über die zweite Datenstruktur zurückverfolgt, welches die abhängigen Vorgängerzustände waren. Die umgekehrte Ausgabe dieser Vorgängerzustände ergibt die Folge von Zuständen mit maximaler Wahrscheinlichkeit, welche die Emissionssequenz erzeugt. Es gilt:

$$i_k = \phi_{k+1}(i_{k+1}).$$

Der so entstandene Pfad heißt Viterbi-Pfad. Bei n Symbolen und m Zuständen beträgt die Laufzeit des Viterbi-Algorithmus $O(nm^2)$ mit Speicherverbrauch $O(nm)$.

2.4.2 Der Baum-Welch-Algorithmus

Der Baum-Welch-Algorithmus löst das Trainingsproblem (Fink [Fin03]). Er wird eingesetzt, um ein HMM so zu trainieren, dass die Emissionswahrscheinlichkeit des Modells für eine gegebene Sequenz maximiert wird. Hierbei werden die Emissionswahrscheinlichkeiten für die Zeichen der einzelnen Zustände und die Start- und die Transitionswahrscheinlichkeiten zwischen den Zuständen optimiert. Der Baum-Welch-Algorithmus ist ein Spezialfall des Expectation-Maximization-Algorithmus (EM-Algorithmus), der iterativ einen Maximum-Likelihood-Schätzwert ermittelt. Diese beiden Verfahren werden in den folgenden Abschnitten erklärt.

2.4.2.1 Maximum-Likelihood-Methode

Sei ein Phänomen durch eine statistische Verteilung ausreichend gut beschrieben. Bei einer gegebenen Datenmenge, als Ergebnis eines Zufallsexperiments interpretiert, können die Parameter der zu Grunde liegenden angenommenen Verteilung mit Hilfe der Maximum-Likelihood-Methode (ML-Methode) geschätzt werden. Die ML-Methode basiert darauf, dass für eine gegebene Menge beobachteter Daten (genannt Realisierung) die unbekannten Parameter der Verteilung so bestimmt werden, dass diese den Daten eine maximale Wahrscheinlichkeit zuordnet. Ein Schätzverfahren $T_n(X) : \mathbb{X} \to \Theta$, bei dem für alle Realisierungen $X \in \mathbb{X}$ auch Schätzwerte existieren, heißt Maximum-Likelihood-Schätzer.

$$L(\theta|X) := \prod_{x \in X} P(x|\theta)$$

heißt Likelihood-Funktion, wobei $X = x_1, ..., x_n$ eine Realisierung und $P(x|\theta)$ die Dichtefunktion von X bzgl. einer Menge von Parametern $\theta \in \Theta$ darstellt. Ein Parameterwert $\hat{\theta} = \hat{\theta}(X)$ heißt Maximum-Likelihood-Schätzwert, wenn für alle θ gilt:

$$L(\hat{\theta}|X) \geq L(\theta|X).$$

2.4.2.2 Expectation-Maximization-Algorithmus

Der EM-Algorithmus ist eine Methode, einen Maximum-Likelihood-Schätzwert iterativ zu bestimmen. Hierbei sei wieder eine Realisierung X gegeben, von der angenommen wird, dass sie von einer Wahrscheinlichkeitsverteilung erzeugt wurde, deren Parameter jedoch unbekannt sind. Ziel ist es, diese Parameter so zu bestimmen, dass die Verteilung die Daten optimal beschreibt. Dies geschieht in zwei Schritten:

1. E-Schritt (Expectation)
2. M-Schritt (Maximization)

Zu Beginn werden die unbekannten Parameter per „educated guess“ oder zufällig bestimmt. Dann folgt eine Schleife, in der jeweils zuerst der E-Schritt und dann der M-Schritt durchgeführt wird. Im E-Schritt wird zunächst der Erwartungswert der Daten bestimmt. Im M-Schritt werden diese Parameter so verändert, dass der Erwartungswert nach der Anpassung größer ist als vor der Anpassung. Diese Schritte werden wiederholt bis keine bzw. nur noch eine sehr geringe Verbesserung erreicht wird. Jede Iteration sorgt also für eine Erhöhung der Likelihood-Funktion und konvergiert insgesamt gegen ein lokales Maximum der Likelihood-Funktion.

2.4.2.3 Baum-Welch-Schleife

Der Baum-Welch-Algorithmus dient der Bestimmung der Parameter eines Hidden-Markov-Modell. Er trainiert ein Hidden-Markov-Modell so (siehe Kapitel 2.4), dass die Emissionswahrscheinlichkeit für eine gegebene Sequenz optimiert wird. Hierbei werden die folgenden drei Phasen durchlaufen.

2.4.2.4 Forward-Phase

In der Forward-Phase werden die Hilfsvariablen $\alpha_i(k)$ berechnet, die die Wahrscheinlichkeit angeben, sich bei der Erzeugung der gegebenen Sequenz zum Zeitpunkt k in Zustand s_i zu befinden. Die Wahrscheinlichkeit, sich zum Zeitpunkt 1 in Zustand s_i zu befinden, ist:

$$\alpha_i(1) = \pi_i e_{s_i,x_1}.$$

Die Wahrscheinlichkeit, zum Zeitpunkt $k+1$ in Zustand s_j zu sein, ist:

$$\alpha_j(k+1) = \left(\sum_{i=1}^{|S|} \alpha_i(k) t_{i,j} \right) e_{s_j,x_{k+1}}.$$

Die Wahrscheinlichkeit, die Sequenz X unter der HMM-Konfiguration θ zu emittieren, ist:

$$P(X|\theta) = \sum_{i=1}^{|S|} \alpha_i(l),$$

wobei $l = |X|$ gilt, also l den Zeitpunkt der letzten Iteration beschreibt.

2.4.2.5 Backward-Phase

In der Backward-Phase werden die Hilfsvariablen $\beta_i(k)$ berechnet. Dabei gibt $\beta_i(k)$ die Wahrscheinlichkeit an, die Restsequenz $x_{k+1}, ..., x_l$ zu erzeugen, falls zum Zeitpunkt k der Zustand s_i erreicht wird. Die Wahrscheinlichkeit für eine leere Restsequenz zum Zeitpunkt k ist für alle Zustände gleich 1:

$$\beta_i(k) = 1.$$

Ausgehend von Zustand s_i ist die Wahrscheinlichkeit für die Restsequenz $x_{k+1}, ..., x_l$ die Summe über alle Zustände s_j der Transitionswahrscheinlichkeiten von Zustand s_i in den Zustand s_j multipliziert mit der Emissionswahrscheinlichkeit des Buchstabens x_{k+1} in Zustand s_j, multipliziert mit der Wahrscheinlichkeit ab Zustand s_j das Wort $x_{k+2}, ..., x_l$ zu erzeugen (ab Zeitpunkt $k+1$):

$$\beta_i(k) = \sum_{j=1}^{|S|} t_{i,j} e_{s_j, x_{k+1}} \beta_j(k+1).$$

Die Wahrscheinlichkeit, das Wort X unter der HMM-Konfiguration θ zu emittieren, ist die Summe über alle Zustände s_i die Wahrscheinlichkeit, das Wort $x_2, ..., x_l$ ab Zustand s_i zu erzeugen, multipliziert mit der Startübergangswahrscheinlichkeit für diesen Zustand multipliziert mit der Emissionswahrscheinlichkeit des Buchstaben x_1 in diesem Zustand:

$$P(X|\theta) = \sum_{i=1}^{|S|} \beta_{s_i}(1) \pi_i e_{s_i, x_1}.$$

2.4.2.6 Setzen der Werte

Zur Berechnung der neuen Transitions- und Emissionswahrscheinlichkeiten werden die Hilfswerte $\gamma_i(k)$ und $\xi_{i,j}(k)$ berechnet. Dabei beschreibt $\gamma_i(k)$ die Wahrscheinlichkeit, unter der aktuellen Konfiguration θ und bei dem zu erzeugenden Wort X, zum Zeitpunkt k im Zustand s_i des HMM zu sein:

$$\gamma_i(k) = \frac{\alpha_i(k)\beta_i(k)}{\sum_{j=1}^{|S|} \alpha_j(k)\beta_j(k)}.$$

Der Wert $\xi_{i,j}(k)$ gibt die Wahrscheinlichkeit an, unter der aktuellen Konfiguration θ und dem zu erzeugenden Wort X, zum Zeitpunkt k in Zustand s_i zu sein und direkt in Zustand s_j über zu gehen:

$$\xi_{i,j}(k) = \frac{\alpha_i(k) t_{i,j} e_{j,x_{k+1}} \beta_j(k+1)}{\sum_{i=1}^{|S|} \sum_{j=1}^{|S|} \alpha_i(k) t_{i,j} e_{j,x_{k+1}} \beta_j(k+1)}.$$

Eine Angabe der erwarteten Besuche in Zustand s_i ist:

$$\sum_{k=1}^{l} \gamma_i(k).$$

Die Anzahl der erwarteten Transitionen von Zustand s_i in Zustand s_j ist:

$$\sum_{k=1}^{l-1} \xi_{i,j}(k).$$

Die neuen Werte für das HMM sind:

$$\begin{aligned} \pi_i &= \gamma_i(1). \\ t_{i,j} &= \frac{\sum_{k=1}^{l-1} \xi_{i,j}(k)}{\sum_{k=1}^{l} \gamma_i(k)}. \\ e_{s_i,\sigma_j} &= \frac{\sum_{k=1,\sigma_j=x_k}^{l} \gamma_i(k)}{\sum_{k=1}^{l} \gamma_i(k)}. \end{aligned}$$

2.4.2.7 Zusammenfassung

Die vom Baum-Welch-Algorithmus berechneten Werte für Transitions- und Emissionswahrscheinlichkeiten konvergieren gegen ein (lokales) Maximum. Dieses ist erreicht, wenn der Algorithmus über mehrere Schritte hinweg nur noch Verbesserungen unterhalb einer gegebenen ε-Schranke erzielt. Wird das globale Maximum benötigt, kann der Algorithmus mit unterschiedlichen Anfangsbelegungen für das HMM mehrmals gestartet werden, um die Wahrscheinlichkeit zu erhöhen, ein globales Maximum zu finden.

2.4.3 Profile HMM

Eine spezielle Form der Hidden Markov Modelle sind die Profile HMM (vgl. Eddy [Edd98]). Sie werden bei der Alignierung mehrerer Sequenzen benutzt. In diesem Kapitel wird ein Überblick über den strukturellen Aufbau und die Funktion der Profile HMM gegeben.

2.4.3.1 Struktureller Aufbau eines Profile HMM

Profile HMM sind so genannte Links-Rechts-HMM, die nur in eine Richtung durchlaufen werden können. Es gibt also keine Transitionen zu einem vorherigen Zustand (siehe Abbildung 2.9). Die Zustandsmenge unterteilt sich in drei Typen, die ein unterschiedliches Emissionsverhalten aufweisen:

Match- und Insert-State: Beim Verlassen dieses Zustandstyps wird gemäß einer Wahrscheinlichkeitsverteilung, die für jeden Match- und Insert-State separat verwaltet wird, ein Symbol eines vorher definierten Alphabetes emittiert. Dieses entspricht in den betrachteten Fällen dem Sequenzalphabet, also zumeist dem DNA- oder Protein-Alphabet.

Delete-State: Beim Verlassen dieses Zustandes wird kein Symbol emittiert. Dementsprechend gibt es keine Emissionswahrscheinlichkeiten.

Jeweils ein Match-, Insert- und Delete-State sind zu einer Spalte zusammengefasst. Vor der ersten Spalte gibt es einen zusätzlichen Insert-State, um Einfügungen am Anfang der Sequenz berücksichtigen zu können. Von jedem Zustand einer Spalte aus gibt es eine Transition zum Match- und zum Delete-State der nächsten Spalte, sowie eine Transition

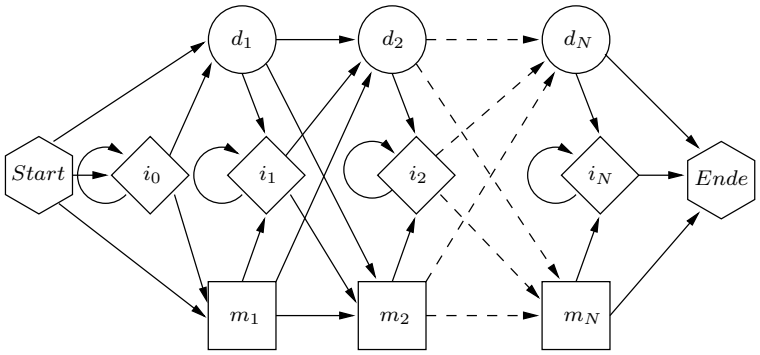

Abbildung 2.9: Die Abbildung stellt eine beispielhafte Struktur eines Profile HMM dar. Grundlegend sind die drei Zustandsarten Match (unten), Insert (mitte) und Delete (oben) in einer Spalte angeordnet. Vor der ersten Spalte existiert ein zusätzlicher Insert-State.

zu dem Insert-State in der selben Spalte (siehe Abbildung 2.9). Es gibt insbesondere eine Transition von jedem Insert-State auf sich selbst.

2.4.3.2 Multiple Sequenzalignments mit Profile HMM

Um mit einem Profile HMM ein Multiples Sequenzalignment (siehe Kapitel 2.3) zu erzeugen, ist zunächst ein Training nötig. Dieses dient dazu die Wahrscheinlichkeit für die Emission einer Sequenz zu erhöhen. Schließlich muss aus dem Profile HMM für jede Sequenz ermittelt werden, an welcher Stelle Leerzeichen eingefügt werden. Dies geschieht anhand der in Kapitel 2.4.1.5 vorgestellten Viterbi-Pfade. Die Viterbi-Pfade der Sequenzen werden parallel durchlaufen und das Alignment nach folgendem Schema erzeugt:

Match-State: Wird bei der Alignierung einer Sequenz im Profile HMM ein Match-State durchlaufen, so wird das nächste Symbol im Multiplen Alignment an der Stelle der entsprechenden Sequenz eingefügt. In Bereichen hoher Ähnlichkeit werden häufig Match-States durchlaufen. Die so erfolgte Einfügung kann zu einem Mismatch führen.

Insert-State: Wird bei der Alignierung einer Sequenz im Profile HMM ein Insert-State durchlaufen, liegt an dieser Stelle eine Einfügung vor. In allen anderen Sequenzen eines Multiplen Alignments, die sich an dieser Stelle nicht in einem Insert-State befinden, muss eine Lücke eingefügt werden. In Bereichen geringer Ähnlichkeit werden häufig Insert-States durchlaufen.

Delete-State: Wird bei der Alignierung für eine Sequenz im Profile HMM ein Delete-State durchlaufen, muss in der Sequenz ein Leerzeichen eingefügt werden, da an dieser Stelle eine Löschung vorliegt.

2.4.4 Generalized Pair HMM

Ein Generalized Hidden Markov Modell (GHMM) ist eine verallgemeinerte Form des HMM, dessen einzelne Zustände statt Einzelsymbolen Symbolketten emittieren können.

Dazu werden die beschriebenen HMM um Verweildauerwahrscheinlichkeiten $d_{ij} \in D$ erweitert, wobei i der Index für den Zustand und j die Emissionslänge ist. Zur Lösung des Dekodierungsproblems werden modifizierte Viterbi-Algorithmen eingesetzt, deren Laufzeiten allerdings so schlecht sind, dass für längere Sequenzen meist approximative Methoden genutzt werden, welche nicht notwendigerweise global optimal sind [Pac02] [CWS01].

Um die Verweildauerwahrscheinlichkeiten der einzelnen Zustände besser verwalten zu können, ist es üblich, diese über eine Menge von Funktionen $f_{s_i} : \mathbb{N} \rightarrow \mathbb{R}$ zu definieren, anstatt eine große Menge von Wahrscheinlichkeiten abspeichern zu müssen. Ein Generalized Pair Hidden Markov Modell (GPHMM) emittiert im Gegensatz zum GHMM in jedem Zustand zwei Symbolketten parallel. So können Wechselbeziehungen zwischen zwei Sequenzen in einem Modell erfasst und für die Analyse berücksichtigt werden.

2.5 Evolutionäre Algorithmen

Evolutionäre Algorithmen sind eine zu den probabilistischen Optimierungsverfahren gehörende Klasse von Algorithmen. Zum Finden und Verbessern von Lösungen für ein gegebenes Problem werden, in Anlehnung an die natürliche Evolution, Lösungen erzeugt und verworfen. Ihr Ursprung lässt sich nicht eindeutig auf eine Person oder einen Zeitpunkt festlegen, da verschiedene Verfahren unabhängig voneinander entwickelt wurden. Fogel entwickelte 1962 die Evolutionäre Programmierung (EP) [Fog62], Holland 1962 die Genetischen Algorithmen (GA) [Hol62] und Rechenberg und Schwefel 1965 Evolutionsstrategien (ES) [Rec65] [Sch65].

Während in der EP als Lösungen für das gegebene Problem endliche Automaten gesucht werden, sind GA und ES auf allgemeinere Probleme mit reelwertigen oder kombinatorischen Lösungen anwendbar. Sie unterscheiden sich insbesondere in der Repräsentation der Lösungen des Problems. Während diese in ES nicht vorgegeben ist, sondern abhängig vom Problem gewählt wird, ist sie bei GA immer ein Vektor von Binärwerten. Es existieren noch weitere Unterschiede zwischen den Algorithmen. Allgemein werden diese und die aus ihnen hervorgegangenen Verfahren unter dem Oberbegriff Evolutionäre Algorithmen (EA) zusammengefasst. Somit umfasst der Begriff EA auch Algorithmen, die sich nicht eindeutig der EP, den GA oder den ES zuordnen lassen.

Nach dem initialen Erzeugen von Lösungen für das jeweilige Problem wird bei jedem EA eine Schleife ausgeführt. Die Lösungen werden als Individuen bezeichnet. Aus der vorhandenen Menge von Individuen (Population) werden anschließend Gruppen definierter Größe ausgesucht, aus welchen durch einen als Rekombination bezeichneten Prozess neue Individuen erzeugt werden. Daraufhin werden die neu erzeugten Individuen durch (Pseudo-) Zufallszahlen verändert. Dies wird als Mutation bezeichnet. Die so erzeugten Individuen werden erneut auf ihre Qualität hin bewertet. Diese Auswertung wird auch als Auswertung der Fitnessfunktion bezeichnet. Die neuen Individuen mit dem besten Fitnesswert werden als neue Population selektiert. Je nach Art des EA werden Individuen der vorangegangenen Generation mit in die Selektion einbezogen. Ist ein definiertes Terminierungskriterium wie zum Beispiel eine vorher festgelegte Anzahl von Schleifendurchläufen erreicht, wird der Algorithmus beendet. Der Ablauf wird in Abbildung 2.10

graphisch dargestellt. Ein nach diesem Schema aufgebauter Algorithmus bietet gegen-

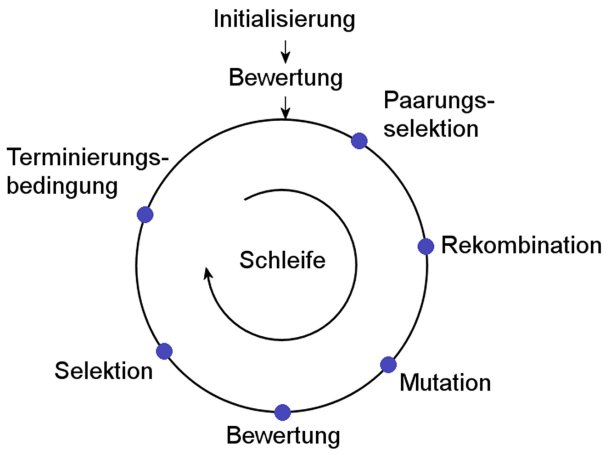

Abbildung 2.10: Nach einer Initialisierung und Bewertung der Population wird folgende Schleife bis zum Erfüllen eines Abbruchkriteriums ausgeführt. Individuen werden zur Rekombination ausgewählt, die Rekombination durchgeführt und die neu erzeugten Individuen mutiert. Anschließend werden sie bewertet und als neue Population für den nächsten Schleifendurchlauf selektiert, daraufhin wird die Terminierungsbedingung überprüft.

über deterministischen Algorithmen, wie beispielsweise gradientenbasierten Verfahren, den Vorteil, dass er bei der Suche im Lösungsraum lokale Extrema überwinden kann und außerdem aufgrund der allgemeinen Formulierung vielseitig und robust anwendbar ist. Diese Eigenschaften sind bei vielen in der Praxis relevanten Problemen sehr vorteilhaft, weshalb EA bei vielen verschiedenen Problemstellungen angewendet werden können. Zur genaueren Beschreibung der einzelnen Schritte werden im Folgenden ES als spezielle Variante der EA anhand des Artikels von Beyer und Schwefel [BS02] genauer erläutert.

2.5.1 Der ES-Basisalgorithmus

Um den Ablauf eines EA genauer beschreiben zu können, sind eine Vielzahl von Parametern notwendig. Für eine ES sind einige davon:

$\boldsymbol{\mu}$: Anzahl Individuen in der Population.

$\boldsymbol{\lambda}$: Anzahl erzeugter Individuen pro Generation.

$\boldsymbol{\rho}$: Anzahl der Individuen, die für das Erzeugen eines neuen Individuums benutzt werden.

$\mathbb{P}_{\boldsymbol{p}}$: Population, aus der neue Individuen erzeugt werden („Elternpopulation“).

$\mathbb{P}_{\boldsymbol{o}}$: Population neu erzeugter Individuen („Nachkommen“).

Wird in der ES $\mathbb{P}_p$ stets durch eine Teilmenge von $\mathbb{P}_o$ ersetzt, so spricht man von einer „Komma-Strategie“. Sofern Individuen aus $\mathbb{P}_p$ auch im nächsten Schleifendurchlauf

existieren, wird von einer „Plus-Strategie“ gesprochen. Als verkürzende Schreibweise wird $(\mu/\rho, \lambda)$ bzw. $(\mu/\rho+\lambda)$ benutzt. Als Verallgemeinerung der Selektionsstrategie kann auch ein Parameter κ eingeführt werden, der die Anzahl der Generationen, die ein Individuum überlebt, angibt. Für diesen Parameter κ gilt, dass $\kappa = 0$ der „Komma-Strategie“ und $\kappa = \infty$ der „Plus-Strategie“ entspricht. Die einzelnen Individuen a_i einer Population lassen sich als

$$a_i := (\vec{y}_i, \vec{s}_i, F(\vec{y}_i)) \tag{2.3}$$

beschreiben, wobei $\vec{y}_i$ die eigentliche Lösung des Problems enthält (auch als Objektparameter bezeichnet), während $\vec{s}_i$ Strategieparameter enthält, die Einfluss auf den Ablauf des Algorithmus, insbesondere die Mutation, nehmen können. Diese Parametereinstellungen können bei verschiedenen Individuen unterschiedlich sein und im Ablauf des Algorithmus verändert werden, weshalb sie auch als endogene Strategieparameter bezeichnet werden. $F(\vec{y}_i)$ in Formel 2.3 ist der Fitnesswert, also die Qualität der von a_i repräsentierten Lösung, der sich durch die Funktion F berechnen lässt. Der Ablauf einer allgemeinen ES wird im Algorithmus 1 in Pseudocode dargestellt. Dabei wählt die marriage-Funktion

$g := 0$;
initialize$\left(\mathbb{P}_p^{(0)} := \left\{\left(\vec{y}_m^{(0)}, \vec{s}_m^{(0)}, F(\vec{y}_m^{(0)})\right), m = 1, \ldots, \mu\right\}\right)$;
repeat
  for $l := 1$ *to* λ **do**
    $\mathbb{E}_l :=$ marriage$\left(\mathbb{P}_p^{(g)}, \rho\right)$;
    $\vec{s}_l :=$ s-recombination$(\mathbb{E}_l)$;
    $\vec{y}_l :=$ y-recombination$(\mathbb{E}_l)$;
    $\tilde{\vec{s}}_l :=$ s-mutation$(\vec{s}_l)$;
    $\tilde{\vec{y}}_l :=$ y-mutation$(\vec{y}_l, \tilde{\vec{s}}_l)$;
    $\tilde{F}_l := F(\tilde{\vec{y}}_l)$;
  end
  $\mathbb{P}_o^{(g)} := \left\{\left(\tilde{\vec{y}}_l, \tilde{\vec{s}}_l, \tilde{F}_l\right), l = 1, \ldots, \lambda\right\}$;
  case *selection-type* $= (\mu, \lambda)$
    $\mathbb{P}_p^{(g+1)} :=$ selection$\left(\mathbb{P}_o^{(g)}, \mu\right)$;
  end
  case *selection-type* $= (\mu + \lambda)$
    $\mathbb{P}_p^{(g+1)} :=$ selection$\left(\mathbb{P}_o^{(g)}, \mathbb{P}_p^{(g)}, \mu\right)$;
  end
  $g := g + 1$;
until *termination-condition* ;

Algorithmus 1 : ES-Basisalgorithmus

ρ Individuen aus $\mathbb{P}_p$ und erzeugt aus diesen durch die Rekombinationsoperationen ein neues Individuum. Die Auswahl der Elternindividuen verläuft dabei zufällig gleichver-

teilt. Das neue Individuum wird nach Anwendung von Mutationen in $\mathbb{P}_o$ eingefügt. Die Realisierung der Mutationsoperatoren sowie die Festlegung der Selektions- und Terminierungskriterien sind dabei nicht festgelegt, sondern sollten problemspezifisch gewählt werden.

2.5.2 Rekombination

Zur Rekombination von Individuen sind zwei Arten von Rekombinationsoperatoren, diskrete und intermediäre Rekombination, gebräuchlich. Bei der intermediären Rekombination werden alle Elternindividuen gleichermaßen berücksichtigt, indem das neue Individuum durch Berechnung der (evtl. gewichteten) Mittelwerte

$$r = \frac{1}{\rho} \sum_{m=1}^{\rho} (x_m)$$

erzeugt wird. Dies ist für reellwertige Suchräume problemlos möglich. In der diskreten Rekombination wird jeder Wert a_i^j eines neu erstellten Individuums a_i aus genau einem Elternindividuum entnommen. Das Elternindividuum wird dabei zufällig aus allen ρ Elternindividuen ausgewählt. Diese Methode findet, aufgrund der Struktur der Individuen, auch in GA Verwendung.

2.5.3 Mutation

Die Funktionsweise eines Mutationsoperators ist problemabhängig. Abgesehen von Restriktionen des Lösungsraumes muss problemspezifisches Wissen in die Entwicklung eines erfolgreichen Operators einfließen. Mutationsoperatoren sollten folgende drei Anforderungen erfüllen:

Erreichbarkeit: Jedes Individuum kann durch eine endliche Anzahl von Anwendungen des Operators zu jedem Punkt des Lösungsraumes mutieren.

Fairness: Die Mutation hat keine Tendenz in Richtung eines Optimums sondern eine Verbesserung des Fitnesswertes ist genau so wahrscheinlich wie eine Verschlechterung.

Skalierbarkeit: Die Mutationsstärke kann angepasst werden, sodass die Individuuen sich bezüglich ihres Fitnesswertes stark oder schwach verändern.

Das Einhalten der genannten Forderungen ist zwar empfehlenswert, aber nicht immer zwingend notwendig. Abhängig von der Problemstellung kann auch ein nicht vollständiges Erfüllen der Forderungen zu guten Ergebnissen der ES führen. Für die in der Mutation verwendeten Zufallszahlen sollte in reelwertigen Lösungsräumen eine Normalverteilung und in ganzzahligen Lösungsräumen eine geometrische Verteilung benutzt werden.

2.5.4 Selektion

Im Unterschied zur Mutation erfolgt die Selektion gerichtet. Es werden immer die Individuen mit den besten Fitnesswerten ausgewählt. Dabei wird zwischen Komma- und Plus-Selektion unterschieden. Da bei der Verwendung der Komma-Selektion die Elterninidividuen verworfen werden, muss hier $\lambda > \mu$ gelten. Der Fall $\lambda = \mu$ würde zu einer rein zufälligen Suche führen. Bei der Plus-Selektion wird immer das beste Individuum beibehalten. Selektionsmethoden mit dieser Eigenschaft werden auch als elitär bezeichnet. Für die Größe von λ gibt es dabei keine allgemeinen Richtlinien oder Einschränkungen. Plus-Selektion wird für diskrete Lösungsräume wie bei kombinatorischen Optimierungsproblemen empfohlen, während Komma-Selektion bei unbeschränkten Lösungsräumen, insbesondere $\mathbb{R}^n$, empfehlenswert ist. Bei allgemeinen EA sind auch andere Selektionsmethoden üblich. So wird beispielsweise eine Turnierselektion benutzt, die zufällig gleichverteilt eine Menge von Individuen auswählt, aus der diejenigen mit dem besten Fitnesswert selektiert werden.

2.5.5 Anpassung von Strategieparametern

Eine ES kann viele Parameter enthalten, die Einfluss auf den Ablauf des Algorithmus haben. Einer der wichtigsten ist die Mutationsstärke σ, welche die Fortschrittsrate φ, also die Geschwindigkeit mit der der Algorithmus gegen ein lokales Optimum konvergiert und die Wahrscheinlichkeit I der Erzeugung eines verbesserten Individuums entscheidend beeinflusst. Dabei gelten die Aussagen 2.4 und 2.5.

$$\sigma \to 0 : I \to 1/2, \varphi \to 0 \tag{2.4}$$

$$\sigma \to \infty : I \to 0, \varphi \to 0 \tag{2.5}$$

Experimente mit einer $(1+1)$-ES und verschiedenen Fitnessfunktionen ergaben, dass φ maximal wird, wenn die gemessenen Werte von I bei ungefähr $1/5$ liegen. Daraus ergibt sich die sogenannte „1/5-Regel“ die besagt, dass für maximales φ die Mutationsstärke so gewählt werden sollte, dass I bei $1/5$ liegt. Ihre Anwendung ist allerdings nur im Spezialfall der $(1+1)$-ES sinnvoll. Sie kann wie in Algorithmus 2 beschrieben zur Anpassung von σ benutzt werden. Neben der 1/5-Regel existieren zahlreiche weitere Verfahren zur

1. $(1+1)$-ES über G Generationen ausführen,
   σ bleibt konstant,
   zähle G_s: Anzahl verbesserter Individuen.
2. $I := G_s/G$
3. $\sigma := \begin{cases} \sigma/c, & wenn\ I > \frac{1}{5} \\ \sigma \cdot c, & wenn\ I < \frac{1}{5} \\ \sigma, & wenn\ I = \frac{1}{5} \end{cases}$ mit eiener Konstanten c: $0.85 \leq c < 1$.
4. Gehe zu 1.

Algorithmus 2 : Die 1/5-Regel

Anpassung von σ und weiterer Strategieparameter, die sich nicht nur auf $(1 + 1)$-ES beschränken. Einige dieser Verfahren werden im Artikel von Beyer und Schwefel [BS02] beschrieben.

2.6 Software

In diesem Abschnitt werden die während der PG Arbeit verwendeten Softwarepakete vorgestellt.

2.6.1 ES-Paket Dortmund

Das vom Lehrstuhl XI der Universität Dortmund entwickelte Paket beinhaltet eine Java-Implementierung einer Standard Evolutionsstrategie (ES) mit Selbstanpassung der Strategieparameter und hält sich dabei an die Empfehlungen der Arbeit von Beyer und Schwefel über ES [BS02].

2.6.1.1 Problemklasse

Die Klasse `Problem` enthält zur Berechnung der Fitnessfunktion die abstrakte Methode `evaluate`. Das ES-Paket Dortmund hält sich an die Einschränkung der ES auf reelwertige Probleme. Die im Basispaket enthaltenen Fitnessfunktionen sind die Ackley-, die Ridge-, die Rosenbrock und die Sphere-Funktion. Diese sind allerdings theoretische Funktionen zur Untersuchung der Optimierungsleistung einer ES.

2.6.1.2 Schnittstelle für Operatoren

Im ES-Paket Dortmund befinden sich einige Variationsoperatoren, die sich für verschiedene Problemstellungen bewährt haben. So ist ein Mutationsoperator vorhanden, der mit Hilfe von normalverteilten Zufallszahlen ein Individuum verändert. Zusätzlich beachtet dieser Operator eine Mutationsstärke pro Individuum, die durch Selbstadaption eingestellt werden können. Zudem ist ein Rekombinationsoperator implementiert, der die für ES typischen Eigenschaften zur Verfügung stellt. So kann über einen Parameter eine diskrete oder eine intermediäre Rekombination gewählt werden. Eine Selektionsklasse für die Plus- und Komma-Methode ist ebenfalls vorhanden. Um eigene Operatoren im Paket nutzen zu können, müssen die Schnittstellen `IMutation`, `IRecombination` und `ISelection` implementiert werden.

2.6.1.3 Komplexe Datenstrukturen

Die zentrale Datenstruktur des ES-Paket Dortmund ist `Population`. Sie enthält eine Menge von `Inividual`-Objekten, die reelwertige Daten speichern können. Um eine komplexere Datenstruktur verwenden zu können, müssen die internen Datenklassen des Optimierungsalgorithmus angepasst werden. Dazu bietet das ES-Paket die Schnittstelle `IIndividual` an. Der Kern des EA wird in der Klasse `ESRun` implementiert. Wenn neu implementierte Operatoren oder Problem-Funktionen hinzugefügt werden müssen, muss

diese Klasse angepasst werden. Da die neue Klasse mit den Schnittstellen kooperieren, ist die Einbettung dieser durch wenige Zeilen Code erbracht. Eine weitere Datenstruktur ist die `ESParameter`-Klasse, welche zur Abfrage der EA-Parameter dient.

2.6.1.4 Die Parameterdatei

Eine Reihe der in Kapitel 2.5.1 benutzten Parameter sowie zusätzlich verwendete Werte werden in der Parameterdatei verwaltet. Sie beinhaltet somit die zentrale Information zur Steuerung der ES. Mit Hilfe der Parameter können beispielsweise

- die Größe der Population durch den Parameter `NPARENTS`,
- die Anzahl zu erzeugender Nachkommen durch den Parameter `NU` (wobei `NU` · `NPARENTS` der Anzahl der Nachkommen entspricht),
- die Lebensdauer der Individuen durch den Parameter `KAPPA` (Individuen können maximal `KAPPA` mal selektiert werden),
- die Anzahl unabhängig durchzuführender Evolutionsdurchläufe durch den Parameter `REPEATS`,
- die Anzahl durchzuführender Auswertungen pro Evolutionsdurchlauf durch den Parameter `ITERATIONS`,
- die Anzahl zu erzeugender Generationen pro Evolutionsdurchlauf durch den Parameter `GENERATIONS`,
- die untere und obere Schranken für den Wert der Größe der Schrittweite durch die Parameter `SHI` und `SLOW`, sowie
- der Parameter `TAUO`, der zur Anpassung der Schrittweite verwendet wird,

festgelegt werden. Einen vollständigen Überblick aller Parameter samt deren Bedeutung gibt eine kommentierte Parameterdatei, `demoSpo.par`, die sich im Verzeichnis `demo` befindet.

2.6.2 BioJava

BioJava ist eine in Java geschriebene Open Source Bibliothek, die den Zugriff auf Datenbanken und Dateien mit biologischen Daten erleichtert und eine Implementierung elementarer Alignierungsalgorithmen wie Smith-Waterman und Needleman-Wunsch (siehe Kapitel 2.3.5 und 2.3.4) enthält. Zudem stellt sie Hidden Markov Modelle mit dem Viterbi- und dem Baum-Welch-Algorithmus (siehe Kapitel 2.4, 2.4.1 und 2.4.2) zur Verfügung. Matthew Pocock und Thomas Down riefen das BioJava-Projekt 1998 ins Leben [PD98], an dem beständig weiterentwickelt wird. Die an der bioinformatischen Forschung beteiligten Institute und Hersteller haben verschiedene Programme entwickelt, die mit unterschiedlichen Formaten arbeiten. Einige davon sind FASTA, EMBL, GenBank und

SwissProt. Ein Framework wie BioJava vereinheitlicht den Zugriff auf die unterschiedlichen Formate durch Schnittstellen. Dadurch kann die Entwicklung von Anwendungen mit Hilfe von BioJava vereinfacht werden. Konkrete Aufgaben von BioJava sind das Lesen und Schreiben von Dateiformaten, einfacher Zugriff auf Sequenzdatenbanken und die Bereitstellung von wichtigen Bioinformatikalgorithmen.

2.6.2.1 Datenstrukturen

Das BioJava-Paket stellt verschiedene Datenstrukturen für biologische Daten zur Verfügung. Die elementarsten Datentypen sind:

Symbole: Symbole sind in BioJava mit Hilfe von Alphabet-Objekten definiert.

Alphabete: Alphabete fassen Symbole zu funktionalen Mengen zusammen. Die gängigen biologischen Alphabete sind DNA, RNA, oder die Aminosäuren der Proteine mit den entsprechenden Symbolen.

Symbollisten: Symbollisten sind Symbolketten, welche die Symbole speichern. Sie ermöglichen das Hinzufügen, Editieren oder Löschen einzelner Symbole.

Sequenzen: Sequenzen sind erweiterte Symbollisten und sind die Hauptdatenstruktur für die meisten enthaltenen Algorithmen.

2.6.2.2 ProfileHMM in BioJava

Im Folgenden soll auf die Struktur von `ProfileHMM` in BioJava eingegangen werden. Ausführlichere Informationen dazu befinden sich in der BioJava-API [PD06].

2.6.2.3 Aufbau

Die wesentlichen Bestandteile eines Profile HMM in BioJava sind die spaltenweise angeordneten Zustände und die den Zustandsübergängen zugeordneten Wahrscheinlichkeiten sowie ein Emissionsalphabet. In den emittierenden Zuständen, also Match- und Insert-States, ist die Wahrscheinlichkeitsverteilung der Emissionen über den einzelnen Symbolen eingebettet. Eine Besonderheit im Vergleich zur formalen Definition (siehe Kapitel 2.4) ist der „magische“ Zustand. Dieser ist praktisch ein stummer Match-State in Spalte Null. Er ist sowohl Start- als auch Endpunkt des Modells, was zum einen die Modellierung der Startwahrscheinlichkeiten für die Implementierung erleichtert und zum anderen erzwingt, dass in einem Profile HMM alle Spalten durchlaufen werden.

2.6.2.4 Konstruktor

Dem Konstruktor der Klasse `ProfileHMM` wird das Emissionsalphabet, die Anzahl der Spalten, eine Verteilung für die Initialisierung der Match- und Insert-States bzw. Delete-States und eine Zeichenkette als Name in Form von Objekten übergeben. Die Anzahl der Spalten lässt sich nach der Instanziierung nicht mehr ändern. Eine nachträgliche

Verlängerung oder Verkürzung ist somit unmöglich. Diese Funktionalität wird jedoch benötigt, um Mutationen für einen EA durchzuführen (vgl. hierzu Kapitel 3.1).

2.6.3 SPOT

Die „Sequential Parameter Optimization Toolbox“ (SPOT) ist ein von Thomas Bartz-Beielstein, Christian Lasarczyk und Mike Preuß am Lehrstuhl XI der Universität Dortmund entwickeltes Software-Paket zur experimentellen Bestimmung von Parametersätzen für Algorithmen. Es ist eine Umsetzung des SPO-Ansatzes aus [BB06] und wurde im Rahmen dieser PG bei verschiedenen Projekten zur Verbesserung der durch die verwendeten EA (siehe Kapitel 2.5) erhaltenen Ergebnisse verwendet. Zur Durchführung als auch zur Auswertung der Experimente ist eine unter [BBL06] frei verfügbare Sammlung von für Matlab [Mat06] geschriebenen Funktionen verwendet worden. Eine genauere Beschreibung des Pakets sowie Hinweise zur Installation und Nutzung befindet sich unter [BBL06]. Im Folgenden werden die Hintergründe zur Methode der SPO kurz erläutert und die Schnittstellen- und Konfigurations-Dateien beschrieben.

2.6.3.1 SPO

Die Methode der sequentiellen Parameteroptimierung versucht ein Verfahren durch sequentielles Testen der einzelnen Parameter, an ein optimales Ergebnis an zu nähern. Dies betrifft insbesondere EA, bei denen es oft eine Vielzahl an Parametern gibt, von denen die Laufzeit und der gefundene Funktionswert essentiell abhängen. Der statistische Hintergrund basiert auf einem Artikel von Sacks [SWMW89]. Dabei werden lineare Regressionsmodelle zur Modellierung der Ergebnisfunktion verwendet. Diese bilden eine Funktion $F : \mathbb{B} \rightarrow \mathbb{E}$ mit dem Parameterraum $\mathbb{B}$ und Ergebnisraum der Funktionsauswertung $\mathbb{E}$ zur Approximation der Abhängigkeit des Algorithmus von seinen Parametern. Für einen Lauf des Algorithmus mit den Parametereinstellungen $\vec{b} \in \mathbb{B}$ und dem Ergebnis $y \in \mathbb{E}$ soll $F(\vec{b}) = y + \epsilon$ mit möglichst kleinem ϵ gelten. So soll mittels weniger Funktionsauswertungen ein Bild gewonnen werden, wo sich im Suchraum gute Parametereinstellungen befinden könnten. Es zeigt sich, dass bei der Optimierung von randomisierten Algorithmen die Hinzunahme eines stochastischen Prozesses das Modell deutlich verbessert werden kann. Dieser stochastische Prozess wird durch einen Mittelwert 0 und eine Kovarianz nach einer zu wählenden Funktion (z.B. der Gauß-Funktion) beschrieben. Dadurch soll die Varianz im Modell geglättet werden.

Zur Erstellung eines solchen Modells ist die Wahl des Designs grundlegend. Der Begriff Design beschreibt hier die Menge der Punkte im Suchraum, sogenannte Designpunkte, auf denen eine Funktionsauswertung durchgeführt werden soll. Diese Menge sollte möglichst gleichverteilt über dem Suchraum gestreut sein. Dabei stellt sich das „Latin Hypercube Sampling“ als intuitiv gutes Verfahren heraus. Dieses Design lässt eine gute Verteilung bei recht wenig Suchpunkten erhoffen. Dabei wird der Suchraum gleichmäßig in Hyperwürfel unterteilt und die Suchpunkte so verteilt, dass in jeder Dimension nur ein Würfel einen Designpunkt beinhaltet. Dies lässt sich nach dem Prinzip „Setze acht Türme so auf einem Schachbrett, dass sie sich nicht gegenseitig angreifen können“ veranschaulichen.

Im eigentlichen Optimierungsprozess wird eine Vorgehensweise von Santner [SWN03] gewählt, welche nach dem Testen des initialen Designs das Modell berechnet und daraus die Wahl von neuen Designpunkten bestimmt. Diese werden zum einen dort angesiedelt, wo das Modell gute Ergebnisse verspricht, zum anderen dort, wo das Modell noch große Unsicherheiten aufweist. So wird das Modell sequentiell verbessert.

2.6.3.2 Konfiguration und Schnittstelle

Eingaben und Ausgaben für SPOT arbeiten Datei basiert. Zentrale Dateien sind:

Projektname.m: Hier werden SPOT eigene Parameter übergeben (Anzahl der SPO-Läufe u.ä.).

Projektname.roi: Hier werden die Namen der Parameter und der Bereich angegeben, in dem optimiert werden soll.

Projektname.des: Datei für die Design Punkte.

Projektname.res: Ergebnisse der Läufe des zu optimierenden Algorithmus.

Der Projektname wird im SPOT-Funktionsaufruf als Parameter übergeben. SPOT erzeugt die Design-Punkte für die zu optimierenden Parameter in Abhängigkeit von den angegebenen Parameterbereichen. Diese werden vom eigentlichen Algorithmus ausgewertet und die Ergebnisse in einer Ergebnisdatei an SPOT zurück gegeben. Daraus werden wiederum neue Design-Punkte erstellt. Um SPOT verwenden zu können, muss der zu optimierende Algorithmus also eine Schnittstelle für das Einlesen der Design-Dateien und die Ausgabe der Ergebnisdateien bereitstellen. Im ES-Paket Dortmund ist diese Funktionalität bereits implementiert.

3 Anwendungsgebiete

3.1 Eigene HMM-Implementierung

Nachdem im ersten Semester der Projektgruppe eine Implementierung von HMM benutzt wurde, die aus dem BioJava-Paket (siehe Kapitel 2.6.2) stammt, wurde im zweiten Semester entschieden, eine eigene Implementierung an zu streben. Ein Grund war der große Speicherbedarf, durch den die Verfahren nicht an echten Probleminstanzen getestet werden konnten. Deshalb sollte der Schwerpunkt der Implementierung auf einer möglichst effizienten Nutzung des Arbeitsspeichers liegen. Eine weitere Motivation bestand darin, dass die Handhabung der speziellen HMM (siehe Kapitel 2.4.3) sowie die Erweiterbarkeit auf weitere HMM-Klassen im BioJava-Paket sehr komplex und ineffizient sind. Schließlich stellte sich bei der neuen Implementierung des Viterbi-Algorithmus heraus, dass die entsprechende Methode aus dem BioJava-Paket für spezielle HMM nicht den korrekten Viterbi-Pfad liefert (siehe Kapitel 3.1.4).

Im Folgenden werden die Ergebnisse der erstellten Implementierung vorgestellt. Dabei wird zunächst die Architektur, dann die wichtigsten Klassen und deren Funktionen beschrieben und abschließend die Effizienz und Korrektheit der Implementierung diskutiert.

3.1.1 Architektur

Bei der Architektur des `HMM`-Pakets steht vor allem die Flexibilität und ein geringer Speicherbedarf im Vordergrund. So sind die Emissionswahrscheinlichkeiten der `HMM` von den Zuständen entkoppelt (siehe Abbildung 3.1) und werden in `Emitter`-Klassen verwaltet. Da der `Emitter` als Java-Interface implementiert wurde, ist es weiterhin möglich, dass Zustände Emissionswahrscheinlichkeiten selbst verwalten. Dies wird beispielsweise im `SingleEmitterState` realisiert. Zudem ist es auch möglich, dass zur Reduzierung des Speicherbedarfs das `HMM` selbst das `Emitter`-Interface implementiert. Die `HMM`-Klasse, die Basisklasse für alle `HMM`-Typen ist, implementiert alle grundlegenden Methoden. Im Gegensatz zur `SimpleHMM`-Klasse sind viele Methoden nur innerhalb des Pakets und in abgeleiteten Klassen sichtbar. Auf diese Weise ist gewährleistet, dass der Benutzer nur Methoden benutzen kann, die für die jeweiligen `HMM`-Unterklassen tatsächlich verwendbar sein sollen. Neben den HMM-Klassen steht dem Benutzer des `HMM`-Pakets auch die `HMMToolbox` zur Verfügung. Diese Klasse enthält, wie in Abbildung 3.2 zu sehen ist, eine Reihe statischer Methoden, die Berechnungen und Manipulationen auf den HMM durchführen. Der Vorteil dieser Klasse besteht darin, dass die Methoden der `HMM`-Klasse uneingeschränkt verfügbar sind und direkt auf ihre Datenstrukturen zugegriffen werden kann. Für die Persistenz der `HMM` (siehe Kapitlel 3.1.4.5) und die Anbindung an BioJava werden die Xerces- ([Apa99]) und die BioJava-Bibliothek benötigt. Neben der eigent-

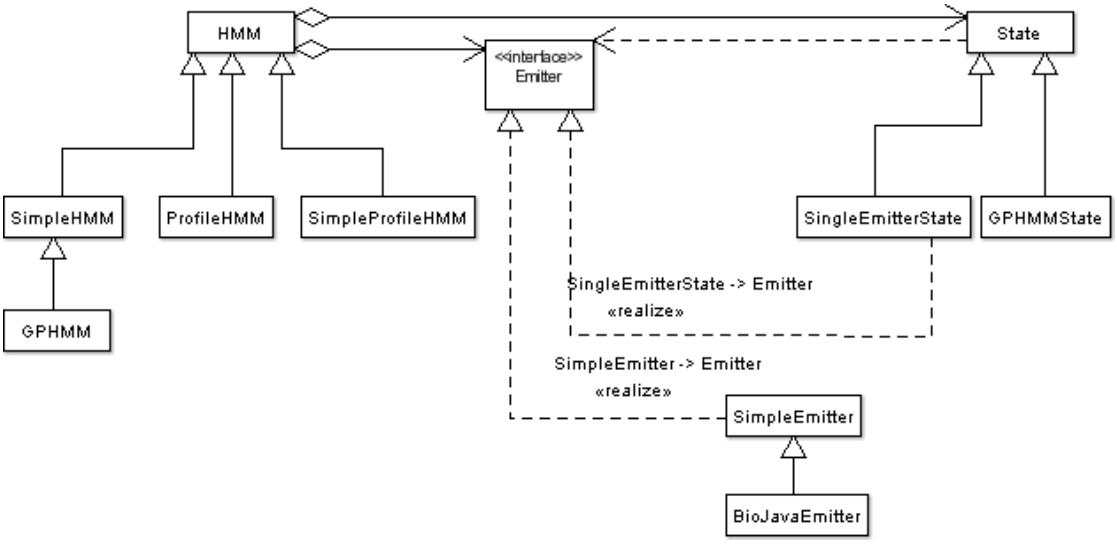

Abbildung 3.1: Ausschnitt des UML-Klassendiagramms des `HMM`-Pakets. Es werden die HMM-Klassen und ihre Vererbungshierarchie dargestellt. Die höchste Abstraktionsebene ist in der ersten Zeile mit den Klassen bzw. Interfaces dargestellt. In den Ebenen darunter befinden sich die konkreten Implementierungen der HMM-Typen und die dazugehörigen Zustände und Emitter.

lichen Implementierung wurden zudem JUnit-Tests und eine API-Dokumentation auf Basis von Javadoc [Sun94] erstellt.

3.1.2 Implementierung der Klasse HMM

Die Klasse `HMM` ist wie beschrieben die Oberklasse aller HMM-Klassen, sie implementiert die gemeinsamen Bausteine aller HMM. Dies sind, wie in Kapitel 2.4 beschrieben, die Zustandsmenge S, die Transitionsmatrix T, das Emissionsalphabet Σ sowie Start- und Emissionswahrscheinlichkeiten Π und E. Hierdurch werden Implementierungen weiterer HMM-Typen vereinfacht und die bereits bestehenden Algorithmen auf sie anwendbar.

3.1.2.1 Datenstrukturen in der Klasse HMM

Während Zustandsmenge S und Emissionsalphabet Σ durch zur Abbildung von Mengen geeignete Datenstrukturen implementiert sind (konkret: `java.util.LinkedList` und `java. util.Set`), werden Transitionsmatrix T sowie Start- und Emissionswahrscheinlichkeiten Π und E in anderer Form in `HMM` repräsentiert als der Definition entsprechenden Konstrukten. Für sie wurden stattdessen spezielle Datenstrukturen implementiert. In `HMM` wird T durch die Klasse `Transition` repräsentiert, wodurch der bei Implementierung einer Matrix zur Anzahl der Zustände quadratische Speicherbedarf vermieden wird. In der `HMM`-Klasse wird nun eine Menge von `Transition`-Objekten verwaltet. Eine weitere Möglichkeit der Implementierung wird in Kapitel 3.1.5 vorgestellt. Auch Π wird in `HMM` in anderer Form als dem der Definition entsprechenden Vektor von Wahrschein-

Abbildung 3.2: Ausschnitt des UML-Klassendiagramms des `HMM`-Pakets. Es wird die `HMMToolbox`-Klasse mit allen für den Benutzer sichtbaren Methoden dargestellt.

lichkeiten implementiert. Jedes HMM enthält stattdessen einen magischen Zustand ohne Emissionen. Dieser Zustand wird durch den Konstruktor erzeugt. Er ersetzt den Vektor Π, indem eine von Null verschiedene Startwahrscheinlichkeit eines Zustands durch eine Transition gleicher Wahrscheinlichkeit vom magischen Zustand aus zum betreffenden Zustand gesetzt wird. Somit können Startwahrscheinlichkeiten ohne die Benutzung neuer Datenstrukturen gespeichert werden. Außerdem können Zustände des HMM Transitionen zum magischen Zustand besitzen, um endlich lange Pfade im HMM zu ermöglichen. Diese „terminierende“ Eigenschaft des magischen Zustands ist beispielsweise in der Klasse `ProfileHMM` (siehe Kapitel 3.1.3) notwendig. Die Emissionswahrscheinlichkeiten E werden nicht direkt im `HMM` verwaltet, sondern nur als Referenzen auf `Emitter`-Objekte, welche im Kapitel 3.1.1 vorgestellt wurden. Deshalb müssen Emitter beim Hinzufügen und Entfernen aus dem HMM explizit an- bzw. abgemeldet werden.

3.1.2.2 Methoden der Klasse HMM

Zu einer Implementierung eines HMM gehören neben den benötigten Datenstrukturen auch Methoden zum Hinzufügen, Entfernen und Verändern von Zuständen, Transitionen

und Emissionen. Diese Methoden sind im `HMM`-Paket in den `HMM`-, `Emitter`- und `State`-Klassen implementiert. Um dem Benutzer eine freie Veränderung der Transitions- und Emissionswahrscheinlichkeiten zu ermöglichen, werden die Wahrscheinlichkeiten nicht durch die Klasse normalisiert. Zusätzlich ist es möglich, durch Aufrufen der Methoden `getSuccessors(State from)` und `getPredecessors(State to)` zu jedem Zustand die Vorgänger- und Nachfolgerzustände zu bestimmen.

3.1.2.3 Beispiel zur Verwendung der Klasse HMM

In diesem Abschnitt wird beispielhaft ein HMM mit drei emittierenden Zuständen erstellt. Zunächst wird dazu ein Objekt der Klasse `SimpleHMM` erzeugt. Als Parameter werden ein Name und ein Emissionsalphabet übergeben. Das Emissionsalphabet kann anstelle der hier benutzten Strings auch Objekte einer beliebigen anderen Klasse enthalten (siehe Listing 3.1). Das neu erstellte HMM ist leer, da noch keine Zustände hinzugefügt wurden. Neue Zustände können durch die Methode `addState(State s)` in das

```
HashSet<String> alphabet = HashSet<String>();
String a = new String("Adenin");
alphabet.add(a);
String g = new String("Guanin");
alphabet.add(g);
String c = new String("Cytosin");
alphabet.add(c);
String t = new String("Thymin");
alphabet.add(t);

SimpleHMM hmm = new SimpleHMM("Demo-HMM", alphabet);
```

Listing 3.1: Erstellen eines Alphabets und eines SimpleHMM-Objekts.

```
State s1 = new State("s1");
State s2 = new State("s2");
State s3 = new State("s3");
try{
    hmm.addState(s1);
    hmm.addState(s2);
    hmm.addState(s3);
}
catch(IllegalEmissionException e){
      e.printStackTrace();
}
```

Listing 3.2: Erstellen von Zuständen und Emittern.

HMM eingefügt werden. Im Beispiel werden drei Zustände zum HMM hinzugefügt (siehe Listing 3.2). Für emittierende Zustände muss weiterhin zumindest ein `Emitter`-Objekt erstellt und zugewiesen werden. Anschließend werden die Emissionswahrscheinlichkeiten gesetzt und der Emitter beim HMM registriert (siehe Listing 3.3). Beim abschließenden Hinzufügen von Transitionen muss beachtet werden, dass auch Transitionen vom und ggf. zum magischen Zustand erstellt werden (siehe Listing 3.4). Das erstellte HMM wird in Abbildung 3.3 dargestellt.

```
SimpleEmitter emitter = new SimpleEmitter();
try{
   s1.registerEmitter(emitter);
   s2.registerEmitter(emitter);
   s3.registerEmitter(emitter);

   emitter.setWeight(alphabet, a, 0.25);
   emitter.setWeight(alphabet, g, 0.25);
   emitter.setWeight(alphabet, c, 0.25);
   emitter.setWeight(alphabet, t, 0.25);

   hmm.registerEmitter(emitter);
}
catch (IllegalEmissionException e){
   e.printStackTrace();
}
```

Listing 3.3: Setzen von Emissionswahrscheinlichkeiten und Registrieren des Emitters.

```
try {
       hmm.addTransition(hmm.getMagicalState(), s1, 1.0);
       hmm.addTransition(s1, s2, 0.7);
       hmm.addTransition(s1, s3, 0.3);
       hmm.addTransition(s2, hmm.getMagicalState(), 1.0);
       hmm.addTransition(s3, hmm.getMagicalState(), 1.0);
}
catch (StateNotFoundException e) {
    e.printStackTrace();
}
```

Listing 3.4: Hinzufügen von Transitionen zum HMM.

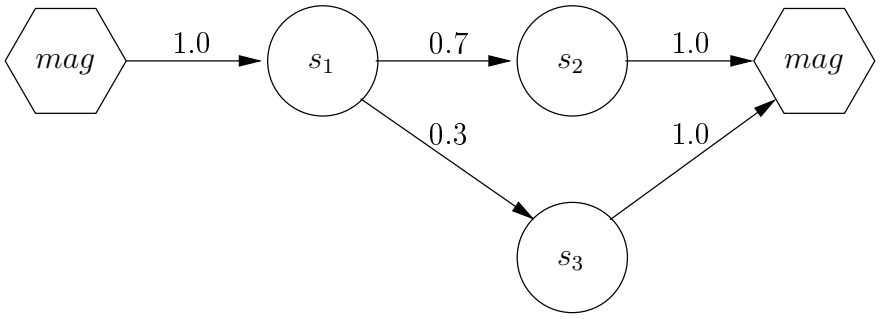

Abbildung 3.3: Graphische Darstellung des in den Listings 3.1 bis 3.4 erstellten `SimpleHMM`-Objekts.

3.1.3 Implementierung von Profile HMM

Ein Spezialfall der HMM, der insbesondere bei der Sequenzerkennung von Bedeutung ist, sind die in Kapitel 2.4.3 beschriebenen Profile HMM. Diese wurden wie in Kapitel 3.3 beschrieben benötigt und eine eigene Klasse `ProfileHMM` implementiert. Die Klasse ermöglicht dem Benutzer eine komfortable Benutzung von Profile HMM und gewährleistet gleichzeitig eine einheitliche und intern konsistente Struktur. Im Folgenden werden die wichtigsten der implementierten Methoden und Datenstrukturen in Funktion und Funktionsweise beschrieben.

3.1.3.1 Datenstrukturen in der Klasse ProfileHMM

Gegenüber einem Objekt vom Typ `HMM` werden in einem `ProfileHMM`-Objekt zusätzliche Daten gespeichert. Grund dafür ist die Struktur eines Profile HMM, die, wie in Kapitel 2.4.3 beschrieben wird, Zustände in die (disjunkten) Mengen Insert, Match und Delete einteilt. Tripel von jeweils einem Zustand einer Klasse bilden eine Spalte des Profile HMM. Die Zugehörigkeit eines Zustands zu einer Klasse wird in `ProfileHMM` durch die Speicherung in einem korrespondierenden `Vector<State>` abgebildet, die Zuordnung zu einer Spalte geschieht über den Index des Zustands im Vektor. Beide Informationen sind auch im Namen eines Zustands enthalten. Da die nullte Spalte nur einen Zustand der Klasse Insert enthält, beinhaltet der Vektor der Insert-Zustände einen zusätzlichen Zustand, wodurch der Index hier um eine Stelle verschoben ist. Neben der Speicherung von Zuständen in Vektoren wird noch eine Variable vom Typ `int` verwaltet, die die aktuelle Anzahl der Spalten speichert.

3.1.3.2 Konstruktor der Klasse ProfileHMM

Der Konstruktor der Klasse erzeugt ein `HMM`-Objekt, das der Struktur eines Profile HMM entspricht. Dabei werden dem Konstruktor neben einem String, der eine Bezeichnung des HMM beinhaltet, eine Anzahl von Spalten sowie ein Emissionsalphabet übergeben. Es ist zu beachten, dass jedem erstellten emittierenden Zustand, also den Match- und Insert-Zuständen, die gleiche Instanz des Alphabets zugeordnet wird. Die Emissions-

wahrscheinlichkeiten werden gleichverteilt gesetzt. Auch die Initialisierung der Transitionswahrscheinlichkeiten geschieht gleichverteilt.

3.1.3.3 Die Methode deleteColumn

Um die Struktur eines Profile HMM nicht zu zerstören, können keine einzelnen Zustände, sondern nur komplette Spalten mit der Methode `deleteColumn(int index)` aus dem HMM entfernt werden. Dabei ist es nicht möglich, die einzige oder die nullte Spalte zu löschen. Durch die Methode werden neben den Zuständen auch die von ihnen ausgehenden Transitionen entfernt. Die Transitionen auf die zu löschende Spalte erhalten die Spalte mit nächstgrößerem Index als neues Ziel. Da sich durch das Löschen der letzten Spalte die Anzahl der von der vorletzten Spalte ausgehenden Transitionen ändert, müssen ihre Wahrscheinlichkeiten modifiziert werden, damit sich die von einem Zustand ausgehenden Transitionen zu 1 aufsummieren. Da nicht klar ist, wie die neuen Transitionswahrscheinlichkeiten gewichtet werden müssen, werden sie gleichverteilt gesetzt.

3.1.3.4 Methoden zum Einfügen von Spalten

Analog zum Entfernen von Spalten ist auch das Einfügen von Spalten möglich. Die Stelle, an der die Änderung im Profile HMM geschehen soll, wird durch den Index der Spalte bestimmt. Dabei werden die Indizes ggf. an ihre neue Position angepasst. Die neue Spalte wird nicht übergeben, sondern neu erstellt. Dabei wird auch hier eine Spalte als Tripel von Zuständen je einer der Klassen Insert, Match und Delete inklusive der von ihnen ausgehenden Transitionen erzeugt. Die neuen Emissions- und Transitionswahrscheinlichkeiten werden gleichverteilt gesetzt. Da die Anzahl ausgehender Transitionen davon abhängt, ob die Spalte die letzte Spalte im Profile HMM ist oder nicht, enthält die Klasse `ProfileHMM` zwei verschiedene Methoden zum Einfügen von Spalten: `insertColumn(int index)` und `appendColumn()`. Die Transitionen auf die neue Spalte sind diejenigen, die die Spalte erreichten, die sich zuvor an dieser Stelle des Profile HMM befand. Bei Benutzung der Methode `appendColumn()` tritt dabei der Sonderfall auf, dass die zuvor letzte Spalte des Profile HMM nicht mehr die letzte Spalte ist. Die Anzahl ausgehender Transitionen muss also geändert werden, ihre Wahrscheinlichkeiten werden gleichverteilt gesetzt. Die Benutzung der Methode `appendColumn()` hat damit nicht nur Auswirkungen auf die neu eingefügte Spalte.

3.1.3.5 Methoden zum Austauschen von Zuständen

Neben der Manipulation des Profile HMM auf Ebene von Spalten ist auch eine Manipulation einzelner Zustände möglich. Um dabei die Konsistenz des Modells zu gewährleisten, ist jedoch nur ein Austauschen von Zuständen realisiert. Hierzu enthält die Klasse `ProfileHMM` die Methoden `setInsertState(int index, State state)`, `setMatchState(int index, State state)` und `setDeleteState(int index, State state)`. Dabei werden über den Index die Position des auszutauschenden Zustands sowie der ersetzende Zustand übergeben. Die Methode prüft den Zustand auf Gültigkeit für seine Position,

also die Anzahl seiner `Emitter` und ggf. das Emissionsalphabet, benennt ihn entsprechend seiner Position im Profile HMM und tauscht ihn anschließend aus. Eingehende und ausgehende Transitionen werden dabei vom ausgetauschten Zustand übernommen.

3.1.4 HMMToolbox

Als eine für den Benutzer des `HMM`-Pakets zentrale Klasse umfasst die `HMMToolbox` eine Sammlung von statischen Methoden, die spezielle Algorithmen auf einem oder mehreren `HMM` durchführen. Ein wichtiges Ziel bei der Entwicklung dieser Klasse war die Integration von Basisfunktionen, die den Umgang mit den `HMM` in Hinblick auf Mutationsoperatoren (siehe Kapitel 2.5.3) erleichtern. Da die `HMMToolbox` ein Teil des `HMM`-Pakets ist, stehen für die Methoden dieser Klasse auch die als `protected` sichtbaren Methoden der Basisklasse `HMM` zur Verfügung. Da die `HMMToolbox` eine Schnittstelle für die Arbeit der gesamten Projektgruppe ist, enthält sie spezifische Methoden für die einzelnen Implementierungen. Im Folgenden wird die Implementierung des Viterbi-, Forward-, Backward- und Baum-Welch-Algorithmus beschrieben und anschließend die speziell für das MSA-Problem (siehe Kapitel 3.3) implementierten Methoden erläutert. Abschließend wird die Implementierung zur `HMM`-Persistenz erklärt.

3.1.4.1 Viterbi-Algorithmus

Wie bereits in Kapitel 2.4.1 beschrieben, berechnet der Viterbi-Algorithmus zu einem gegebenen `HMM` und einer Sequenz eine bestimmte Zustandsfolge. Für diese Zustandsfolge ist die Wahrscheinlichkeit maximal, im `HMM` durchlaufen zu werden und dabei die gegebene Sequenz zu emittieren. Eine solche Zustandsfolge wird Viterbi-Pfad genannt.

Der Standard-Algorithmus zur Berechnung eines Viterbi-Pfades folgt dem Prinzip der Dynamischen Programmierung (siehe Kapitel 2.3.3). Dieses Paradigma ist jedoch für die Implementierung des Viterbi-Algorithmus auf allgemeine `HMM` nicht direkt anwendbar, insbesondere wenn sie nicht-emittierende Zustände enthalten. Solche nicht-emittierenden Zustände werden „stumm“ genannt und dienen, wenn man den Kehrwert der Wahrscheinlichkeiten als Distanzen auffasst, als Abkürzungen in einem Viterbi-Pfad (siehe Abbildung 3.4). Bei einer gegebenen Sequenz gibt es durch die Hinzunahme von stummen Zuständen neue Wege im `HMM`. Diese neuen Wege können eine größere Wahrscheinlichkeit aufweisen, die gegebene Sequenz zu emittieren. Da stumme Zustände nichts emittieren, geht die direkte Zuordnung zwischen Sequenz- und Pfadlänge verloren. Somit ist unklar, wie der für den DP-Ansatz benötigte Wert $\delta_i(j,k)$ (siehe Kapitel 2.4.1) berechnet werden kann. Dieser gibt die größtmögliche Wahrscheinlichkeit an, die Teilsequenz bis Position i der Sequenz X emittiert und die Transition von Zustand s_j nach s_k zuletzt durchgeführt zu haben. Für den Fall, dass weder s_j noch s_k ein stummer Zustand ist, kann man wie gewohnt die Formel

$$\delta_i(j,k) = \delta_{i-1}(j) \cdot t_{j,k} \cdot e_{s_k,x_i}$$

ansetzen. Dabei gibt $t_{j,k}$ die Transitionswahrscheinlichkeit von s_j nach s_k, e_{s_k,x_i} die Emissionswahrscheinlichkeit des Symbols x_i in Zustand s_k im gegebenen `HMM` und x_i das i-te Symbol der gegebenen Sequenz an. Die Transitionswahrscheinlichkeit $t_{j,k}$ bei

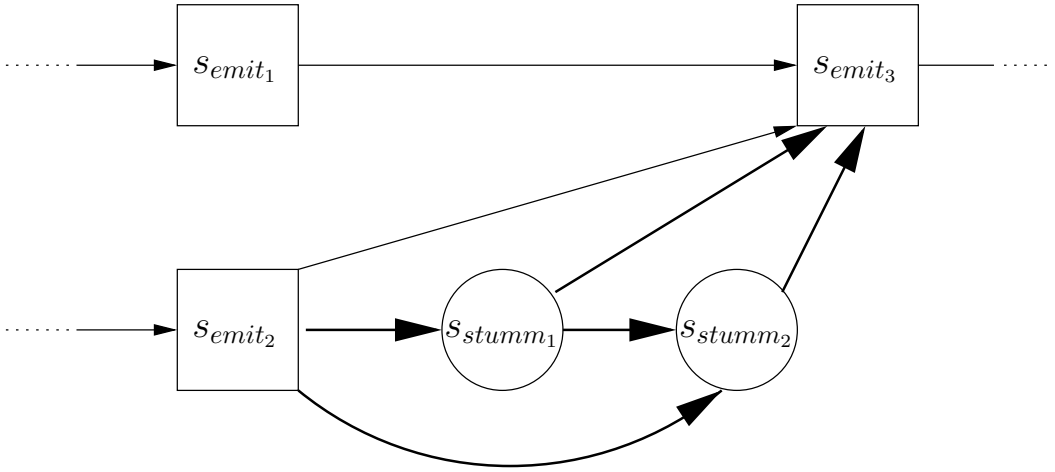

Abbildung 3.4: Die Abbildung zeigt einen Ausschnitt eines HMM mit Zuständen (s_{stumm_1}, s_{stumm_2}) und emittierenden Zuständen (s_{emit_1}, s_{emit_2}, s_{emit_3}). Durch die stummen Zustände existieren Pfade, bei denen Zustände erreicht werden, die ohne das Emissionsverhalten zu ändern auch übersprungen werden können. Einer der längeren Wege, hier mit dicken Pfeilen dargestellt, kann aber in Bezug auf die Pfadwahrscheinlichkeit günstiger sein, als der direkte Weg.

einem `HMM` mit stummen Zuständen ist nicht mehr durch eine einzelne Transition, sondern möglicherweise durch einen ganzen Weg über stumme Zustände gegeben. Eine solche Teilstrecke lässt sich wie folgt berechnen:

$$\delta_i(j,k) = \delta_{i-1}(j) \cdot \max_{\vec{a} \in M^*} (t_{j,a_1} \cdot t_{\vec{a}} \cdot t_{a_n,k}) \cdot e_{s_k,x_i}.$$

Dabei ist M die Menge der stummen Zustände, M^* der Kleensche Abschluss von M und $\vec{a}$ enthält alle möglichen stummen Zustandsfolgen. Außerdem gibt $t_{\vec{a}}$ die Pfadwahrscheinlichkeit des Pfades $(a_1 \to a_2 \to \ldots \to a_n)$ mit $\vec{a} = (a_1, a_2, \ldots, a_n)$ an. Der Viterbi-Algorithmus hat in seiner ursprünglichen Form, also mit $|M^*| = O(1)$, eine Laufzeit von $O(n \cdot m^2)$, wobei n die Länge der Sequenz und m die Anzahl der Zustände des `HMM` ist. Da $\max_{\vec{a} \in M^*}(t_{j,a_1} \cdot t_{\vec{a}} \cdot t_{a_n,k})$ für jedes Symbol der Sequenz und für alle emittierenden Zustandskombinationen (s_j, s_k) berücksichtigt werden muss, ergibt sich eine Worst-Case-Laufzeit von $O(n \cdot m^2 \cdot |M^*|)$, wobei m hierbei auf die Anzahl der emittierenden Zustände beschränkt ist. Da für $M > 0$ der transitive Abschluss M^* unendlich groß ist, gilt $|M^*| = \infty$. Um bei der Implementierung des allgemeineren Viterbi-Algorithmus unter Berücksichtigung von stummen Zuständen eine gute Laufzeit zu erzielen, gibt es grundsätzlich zwei Ansätze:

1. **Approximierung:** Es werden Pfade mit nicht optimalen Pfadwahrscheinlichkeiten als Viterbi-Pfade akzeptiert. Dafür gibt es folgende Möglichkeiten:

   a) Stumme Zustände werden entweder als emittierende Zustände behandelt und eine feste Emissionswahrscheinlichkeit für jedes Symbol verrechnet oder igno-

riert. Dabei muss beachtet werden, dass der berechnete Pfad die Sequenz vollständig emittiert.

b) Zunächst wird ein beliebiger, zulässiger Pfad gewählt und dieser iterativ optimiert.

c) Es wird eine allgemeine (randomisierte) Suchheuristik angewendet, die unter allen zulässigen Pfaden die beste Lösung sucht.

2. **Beschränkung auf tatsächlich benötigte Zusatzinformation:** Der Kehrwert der Transitionswahrscheinlichkeit (t^{-1}) wird als Distanz interpretiert. Auf dem vom `HMM` induzierten Graphen wird das APSP-Problem (All Pairs Shortest Paths) gelöst (es werden die kürzesten Wege zwischen je zwei Zuständen des `HMM` berechnet [Weg05]). Das Ergebnis wird auf die kürzesten Wege zwischen je zwei emittierenden Zuständen reduziert. Diese Wege werden als direkte Transitionen im normalen Viterbi-Algorithmus verwendet. Die Pfadwahrscheinlichkeiten werden wie beschrieben erst dann berechnet, wenn sie benötigt werden.

Da die Viterbi-Pfade bei den Implementierungen der Projektgruppe in die Fitnessfunktion von Evolutionären Algorithmen eingehen, sollte sie möglichst aussagekräftig und für ein `HMM` und eine Sequenz über mehrere Berechnungen konstant sein. Aus diesem Grund wurden keine approximativen Algorithmen verwendet. Als einzige Alternative bleibt nun noch der zweite Ansatz zu betrachten. Dabei wird ausgenutzt, dass nur die Pfade mit maximaler Pfadwahrscheinlichkeit benötigt werden. Durch die Transformation

$$\max_{\vec{a}\in M^*}(t_{j,a_1}\cdot t_{\vec{a}}\cdot t_{a_n,k}) = \max(t_{j,k}, \max_{\vec{a}\in\{\vec{a'}\in M^*|t_{\vec{a'}}>t_{j,k}\}}(t_{j,a_1}\cdot t_{\vec{a}}\cdot ta_n,k))$$

ist M^* für $t_{j,k} > 0$ auf eine endliche Menge eingeschränkt. Neben der Tatsache, dass die Laufzeit für $t_{j,k} = 0$ noch nicht geeignet abgeschätzt wurde, ist $M' = \{\vec{a'} \in M^* | p_p(\vec{a'}) > t_{j,k}\}$ auch für $tj, k > 0$ weiterhin beliebig groß. Allerdings wird $|M'|$ kleiner, je größer $t_{j,k}$ und je kleiner die Transitionswahrscheinlichkeiten $t_{j',k'}$ für $j', k' \in M$ sind. Es genügt, folgenden Wert zu berechnen:

$$\delta_i(k) = \max_{j\in HMM}(\delta_i(j,k))\cdot e_{s_k,x_i}.$$

Sei nun

$$j'_i := \operatorname{argmax}_{j\in HMM}(\delta_i(j,k))$$

ein emittierender Zustand, von dem aus über einen Pfad $\vec{a}$ der emittierende Zustand s_k erreicht werden kann und dessen Pfadwahrscheinlichkeit multipliziert mit $\delta_{i-1}(j'_i)$ maximal ist. Außerdem sei $\vec{a} = (a_1, \ldots, a_n)$ und $m \leq n$, dann gilt

$$t(a_m, \ldots, a_n)\cdot t_{a_n,k}\cdot e_{s_k,x_i} \geq \delta_i(j'_i,k) = \delta_i(k).$$

Wenn nun eine rückwärts gerichtete Tiefensuche nach s'_{j_i} durchgeführt wird, die bei Zustand s_k startet, kann ein Abbruchkriterium

$$t_{(a_m,\ldots,a_n)}\cdot t_{a_n,k} < \begin{cases} p_{i,r}\cdot\delta_{i-1}(j'_{i,r}) & , \; falls\; p_{i,r}\neq 0 \\ \min(\delta_{i-1}(j)) & , \; sonst \end{cases}$$

formuliert werden. Dabei gibt r die Rekursionstiefe der Tiefensuche und $p_{i,r}$ die beste Pfadwahrscheinlichkeit, die bis zur Rekursionstiefe r gefunden wurde, an. Außerdem ist $s_{j'_{i,r}}$ der zugehörige emittierende Vorgängerzustand. Die so konstruierte Tiefensuche terminiert für $i > 0$ genau dann nicht, wenn

$$\min(\delta_{i-1}(j)) = 0$$

gilt, also das `HMM` die Teilsequenz $x_0, \ldots, x_{i-1}$ nicht emittieren kann. Offensichtlich kann für diesen Fall ohnehin kein Viterbi-Pfad berechnet werden und die Tiefensuche kann vorher abgebrochen werden. Der Fall $i = 0$, also die Initialisierung von $\delta_0(j)$, lässt sich auf das SSSP-Problem (Single Source Shortes Paths, [Weg05]) zurückführen. Dieses Problem lässt sich mit dem Algorithmus von Dijkstra [Weg05] in $O(m + n \log n)$, also im wesentlichen mit quadratischer Laufzeit lösen. Es wird außerdem nur $O(n)$ zusätzlicher Speicher benötigt. Damit ist klar, dass der dominierende Faktor der Laufzeit der Fall $i > 0$ ist. Um eine Abschätzung der Laufzeit zu finden, muss die maximale Rekursionstiefe r abgeschätzt werden, da in jeder Rekursion alle Zustände durchlaufen werden. Hieraus ergibt sich die gesamte Laufzeit von $O(n^{3+r})$. Für den Fall $r = 0$, also z.B. wenn es keine stummen Zustände gibt, beträgt die Viterbi-Laufzeit $O(n^3)$. Der Speicherbedarf hingegen ist sowohl für die Initialisierung als auch für $i > 0$ linear in der Anzahl der Zustände.

Nicht nur die Initialisierung, sondern auch die Berechnung von $\delta_i(k)$ allgemein lässt sich auf das SSSP-Problem zurückführen. Es wäre also mit dem Dijkstra-Algorithmus möglich, eine Laufzeit von $O(n^4)$ und zusätzlichem Speicherbedarf von $O(n^2)$ zu erzielen. Da die einzige in der Arbeit der PG genutzte HMM-Variante, die stumme Zustände verwendet, ein topologisch sehr eingeschränktes Profile HMM ist, muss nicht davon ausgegangen werden, dass in jeder Rekursion alle n Zustände durchlaufen werden. Eine genauere Abschätzung der Laufzeit wäre somit $O(n^2 \cdot \log n \cdot p^r)$, wobei p die durchschnittliche Anzahl an Vorgängern eines Zustandes ist. Für `ProfileHMM` ist $p \leq 3$, die Laufzeit ist damit $O(n^2 \log n \cdot 3^r)$. Wenn r z.B. durch hinreichend wenige oder schlecht erreichbare stumme Zustände ebenfalls stark eingeschränkt werden kann, wird für `ProfileHMM` eine asymptotische Laufzeit von $O(n^3)$ erreicht. Der zusätzliche Speicherbedarf liegt in der Größenordnung von $O(n)$.

Zusammenfassend muss bei der genauen Berechnung des Viterbi-Pfades unter Berücksichtigung von stummen Zuständen ein Kompromiss zwischen Speicher- und Laufzeiteffizienz gefunden werden. Auch wenn die Rekursionstiefe bei Testdurchläufen mit gleichverteilten Transitionswahrscheinlichkeiten sehr klein ist, sind weitere Untersuchungen auch in Hinblick auf eine Pfadapproximierung interessant.

3.1.4.2 Baum-Welch-Training

Wie bereits in Kapitel 2.4.2 beschrieben, wird für das Baum-Welch-Training der Forward- und der Backward-Algorithmus benötigt. Der Forward-Algorithmus berechnet zu einem gegebenen `HMM` und einer Sequenz die Wahrscheinlichkeit, dass das `HMM` diese Sequenz emittiert. Der Backward-Algorithmus berechnet das selbe Ergebnis, läuft aber im Gegensatz zum Forward-Algorithmus rückwärts durch die Sequenz. Beide Algorithmen folgen,

wie der Viterbi-Algorithmus, dem Prinzip der Dynamischen Programmierung (siehe Kapitel 2.3.3). Während beim Viterbi-Algorithmus eine maximale Pfadwahrscheinlichkeit berechnet wird, benötigen der Forward- und der Backward-Algorithmus die Summe der Pfadwahrscheinlichkeiten aller möglichen Pfade

$$f_i(k) = \sum_{j \in HMM} (f_i(j,k)) \cdot e_{s_k,x_i}.$$

Ein weiteres Problem für die Implementierung des Forward- bzw. Backward-Algorithmus stellt die Ungenauigkeit der Gleitkommazahlen bei der Darstellung von Wahrscheinlichkeiten dar. Beim Übergang zu logarithmischen Wahrscheinlichkeiten muss neben der Multiplikation auch die Addition unterstützt werden. Zur Berechnung der Addition dient die Kingsbury-Rayner-Formel [Fin03]:

$$\log(a+b) = \log a \cdot \left(1 + \frac{b}{a}\right) = \log a + \log\left(1 + \frac{b}{a}\right) = \log a + \log\left(1 + e^{\log b - \log a}\right).$$

3.1.4.3 Mix-Methoden

Die Methode `mixProfileHMM(List<ProfileHMM> parents, int[][] bounds, String name)` wurde für einen speziellen Anwendungsfall implementiert. Sie wird als Rekombinationsoperator für `ProfileHMM` wie in Kapitel 3.3 beschrieben benötigt, ist jedoch generell unabhängig von einer Verwendung mit Multiplen Sequenzalignments oder evolutionären Algorithmen. Die Funktion der Methode ist, anhand einer Menge von `ProfileHMM`-Objekten ein neues `ProfileHMM` zu erstellen, dessen Daten von den übergebenen Profile HMM kopiert werden. Dieses erhält dann `name` als Namensstring. Die übergebene Menge von Profile HMM wird in Anlehnung an die Verwendung in einem EA als `parents` bezeichnet und als `List<ProfileHMM>` an die Methode übergeben. Das Kopieren der Daten geschieht dann spaltenweise. Jede Spalte im neu erstellten HMM stimmt in den Wahrscheinlichkeiten der Emissionen und der ausgehenden Transitionen mit einem der HMM aus `parents` überein. Außerdem wird aus jedem HMM in `parents` genau ein zuammenhängendes Segment von Spalten kopiert. Die Grenzen dieser Segmente werden in der Matrix `bounds` angegeben. Dazu muss, wenn ρ die Anzahl der Profile HMM in `parents` bezeichnet (siehe Kapitel 2.5), `bounds` eine $(2 \times \rho)$-Matix sein. In dieser gibt der Eintrag $(1, i)$ an, welche Spalte im i-ten HMM aus `parents` die erste Spalte des i-ten Segments ist. Analog ist der Eintrag $(2, i)$ der Index der letzten aus dem i-ten HMM in das i-te Segment zu kopierenden Spalte (siehe Abbildung 3.5), womit die Reihenfolge der Segmente sich nach der Reihenfolge in `parents` richtet. Eine äquivalente Methode für Objekte vom Typ `SimpleProfileHMM` befindet sich ebenfalls in `HMMToolbox`. Die Benutzung wird anhand des in Abbildung 3.5 dargestellten Beispiels in Listing 3.5 veranschaulicht. Zunächst werden das Emissionsalphabet und zwei `SimpleProfileHMM`-Objekte erstellt, aus denen kopiert werden soll. Dazu werden diese in eine `java.util. LinkedList` eingefügt. In der Matrix `bounds` werden die Grenzen der Segmente so bestimmt, dass die ersten zwei Spalten aus dem ersten HMM der Liste und die letzten zwei Spalten aus dem zweiten kopiert werden.

```
HashSet<Character> alpha = new HashSet<Character>();
alpha.add('A');
alpha.add('G');
alpha.add('C');
alpha.add('T');

SimpleProfileHMM hmmA =
   new SimpleProfileHMM("hmmA", alpha, 3);
SimpleProfileHMM hmmB =
   new SimpleProfileHMM("hmmB", alpha, 4);
LinkedList<SimpleProfileHMM> parents =
   new LinkedList<SimpleProfileHMM>();
parents.add(hmmA);
parents.add(hmmB);

int[][] bounds = {{1,3},{2,4}};

SimpleProfileHMM hmmC = HMMToolbox.mixProfileHMM(
   parents, bounds, "hmmC");
```

Listing 3.5: Beispielhafte Verwendung der Methode `mixProfileHMM`. Es werden ein Alphabet sowie zwei Objekte vom Typ `SimpleProfileHMM` erstellt (`hmmA`, `hmmB`). Durch Verwendung der Methode `mixProfileHMM` und der Matrix `bounds` wird ein neues `SimpleProfileHMM`-Objekt `hmmC` so erstellt, dass es seine Emissions- und Transitionswahrscheinlichkeiten aus den Spalten 1 und 2 von `hmmA` und den Spalten 3 und 4 von `hmmB` erhält.

3.1.4.4 Methode zum Multiplen Sequenzalignment

Die Methode `MSAmaker` in den zwei Varianten

- `public static final Alignment MSAmaker(ProfileHMM phmm, RichSequenceDB sequences)` sowie
- `public static final Alignment MSAmaker(SimpleProfileHMM phmm, RichSequenceDB sequences)`

dient der Erstellung eines Multiplen Alignments der übergebenen Sequenzen durch Verwendung eines Profile HMM. Die Parameter-Klassen `Alignment` sowie `RichSequenceDB` entstammen dem BioJava-Paket [PD98], (siehe Kapitel 2.6.2). Die Erstellung eines Multiplen Alignments durch ein Profile HMM wird unter anderem von Krogh [DEKM99] näher beschrieben. Die Sequenzen werden anhand ihrer Viterbi-Pfade an den Spalten des Profile HMM ausgerichtet. Ziel ist es, die übereinstimmenden und die durch Einfügungen entstandenen Sequenzteile durch hinzufügen von Gap-Symbolen an gleiche Positionen der Sequenzen zu bringen. Im Algorithmus werden zwei Schleifen durchlaufen, die über die Anzahl der Sequenzen iterieren. In der ersten Schleife wird für jede Sequenz der

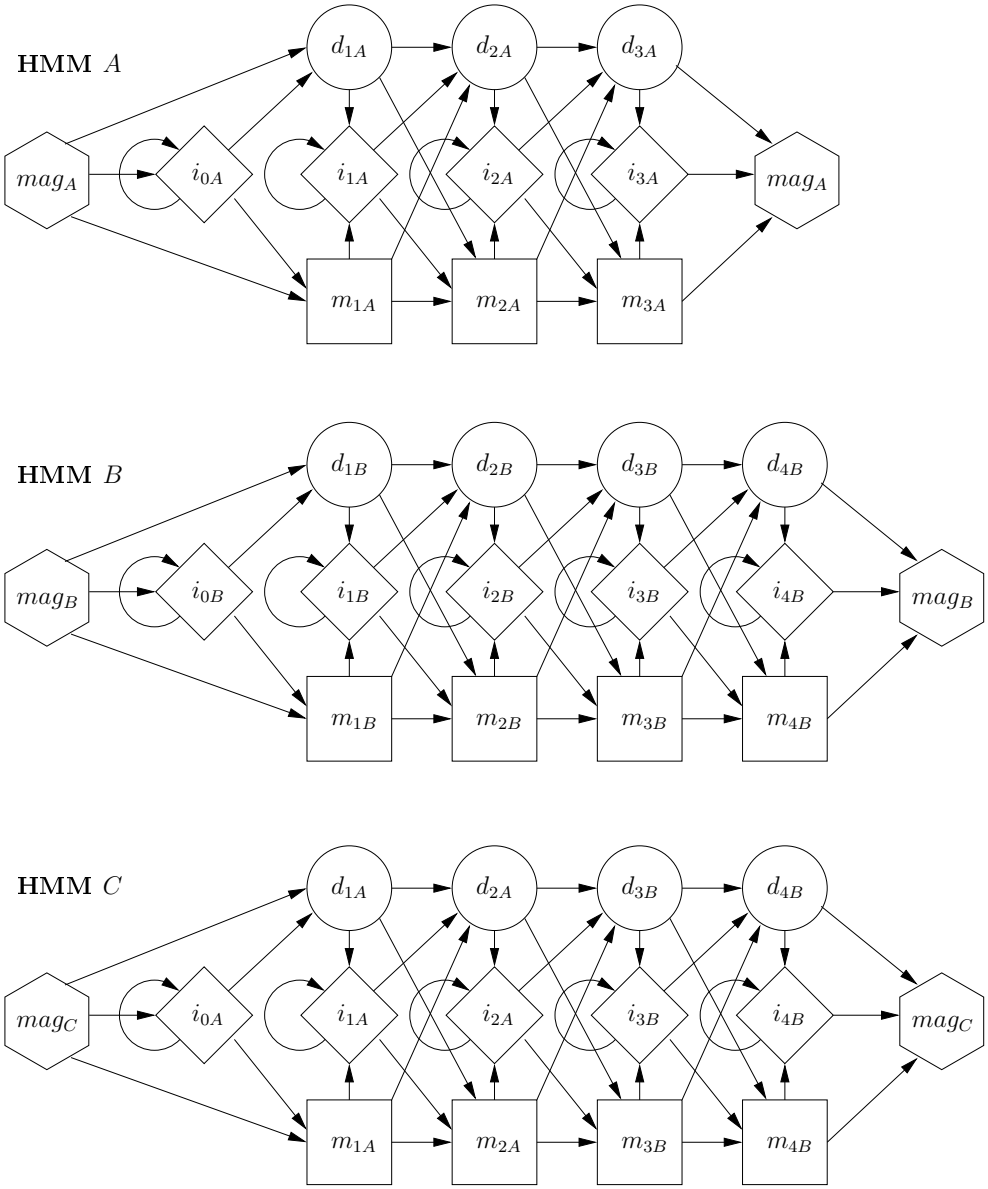

Abbildung 3.5: Die Graphik zeigt die Profile HMM A und B, aus denen das HMM C „gemixt“ wird. HMM C erhält seine Emissions- und Transitionswahrscheinlichkeiten aus den ersten beiden Spalten von A und den letzten beiden von B.

Viterbi-Pfad bestimmt und die Anzahl der pro Spalte emittierten Symbole gezählt (siehe Kapitel 2.4.3). Gleichzeitig zu den Werten für jeden einzelnen Viterbi-Pfad wird der maximale Wert über alle Pfade bestimmt. In der zweiten Schleife werden die Sequenzen mit Gap-Symbolen aufgefüllt. Hierzu wird die Anzahl der Symbole betrachtet, die dem Viterbi-Pfad entsprechend in einer Spalte emittiert werden. Diese Anzahl wird mit dem in der vorangegangenen Schleife bestimmten Maximum für diese Spalte verglichen und die Differenz gebildet. Entsprechend der Differenz werden dann Gap-Symbole an dieser Stelle in die Sequenz eingefügt, so dass sich im Alignment die Symbole gegenüberstehen, die in der gleichen Spalte emittiert wurden.

3.1.4.5 HMM-Persistenz

Für die Reproduzierbarkeit und Analyse der Ergebnisse aus der Projektgruppe ist es nötig, die `HMM`-Objekte zu speichern und zu laden. Da es sich zum Teil um sehr große `HMM` handelt, ist eine übersichtliche Darstellung wünschenswert. Aus diesem Grund wurde eine strukturierte, textuelle Darstellung gewählt. XML (eXtensible Markup Language [BPSM+04]) dient als Grundformat zur Speicherung der Daten. Neben der Speicherung wurde eine Methode zum Laden der XML-Inhalte entwickelt. Ein zusätzliches Java-Interface, das für `HMM`, `State` und `Emitter` und abgeleitete Klassen eingebunden werden kann, ermöglicht es, neben den Basisdaten auch Zusatzinformationen zu speichern und zu laden. Im Folgenden sind die Basisdaten aufgelistet, wobei die fett gedruckten Namen die XML-Elemente bzw. Attribute sind und dahinter eine Beschreibung des Inhalts folgt:

hmm: Das komplette HMM

- **name:** Name des HMM
- **class:** Java-Klasse des HMM
- **id:** Hashcode des HMM-Objekts
- **state:** Einzelne Zustände des HMM
- **transition:** Einzelne Transitionen des HMM

state: Einzelner State

- **name:** Name des Zustandes
- **class:** Java-Klasse des Zustandes
- **id:** Hashcode des Zustandes
- **magical:** „true“, wenn es sich um den magischen Zustand des HMM handelt (siehe Kapitel 3.1.2.1), „false“ sonst
- **mute:** Ist „true“ wenn der Zustand keine Emitter hat, „false“ sonst
- **emitter:** Einzelne Emitter des Zustandes

emitter: Einzelner Emitter

- **class:** Java-Klasse des Emitters
- **id:** Hashcode des Emitter-Objekts
- **emission:** Einzelne Emissionen des Emitters

emission: Einzelne Emission

- **class:** Java-Klasse des emittierten Symbols
- **symbol:** Tatsächliche Ausgabe des Symbols
- **weight:** Emissionswahrscheinlichkeit

transition: Einzelne Transition

- **from:** ID bzw. Hashcode des Zustandes von dem die Transition ausgeht
- **to:** ID bzw. Hashcode des Zustandes zu dem die Transition führt
- **weight:** Transitionswahrscheinlichkeit

Um eine noch übersichtlichere Darstellung zu erzeugen, wurde eine Transformation mittels XSL (eXtensible Stylesheet Language [Ber06]) in ein tabellarisch aufgebautes HTML-Format (Hypertext Markup Language [RHJ99]) definiert. In dieser Darstellung gibt es Tabellen für die Zustände, Transitionen und Emitter (siehe Abbildung 3.6). Die Zustandstabelle listet in einer Zeile die Eigenschaften jeweils eines Zustandes auf. Dabei wird die ID, der Name, die Klasse und eine Liste der Emitter, der Vorgänger- sowie der Nachfolgerzustände in den entsprechenden Spalten aufgeführt. Die Emittertabelle enthält eine Liste der Emissionen mit den zugehörigen Emissionswahrscheinlichkeiten und die Transitionstabelle, die durch Start- und Endzustand eindeutig definierten Transitionen sowie deren Transitionswahrscheinlichkeiten. Zur besseren Navigation kann über Hyperlinks an verschiedene Stellen des Dokumentes gesprungen werden, die entsprechend markiert werden. So ist es z.B. möglich, über die für einen Zustand aufgelisteten Nachfolgezustände direkt zu der zugehörigen Transition zu springen und über die dort aufgelisteten Zustände zu den jeweiligen Zuständen in der Zustandstabelle.

3.1.5 Effizienz der Implementierung

Bei der Verwendung des `HMM`-Pakets wurden die erstellten HMM durch Heuristiken auf verschiedene Ziele hin optimiert. Es bestand die Notwendigkeit, viele unter Umständen große HMM gleichzeitig im Speicher zu halten und viele Operationen auf diesen vorzunehmen. Eine speichereffiziente Verwaltung der HMM sowie eine Manipulation, die nur geringe Rechenzeit beansprucht ist wichtig, um in vertretbarer Zeit Ergebnisse zu erhalten. Die Speichereffizienz ist dabei von höherer Priorität, weil ein Programmlauf bei zu hohem Speicherbedarf entweder unmöglich ist oder langsamer virtueller Speicher verwendet werden muss. Dagegen hat eine Verbesserung der Rechenzeit auf Ebene von einfachen Manipulationen eines HMM geringere Auswirkungen. Zur Messung des Speicherbedarfs

HMM_1 (hmm.SimpleHMM)

Zustände

#	ID	Name	Typ	Emitter	Vorgänger	Nachfolger
1	12115695	Mag	hmm.State			
2	20196260	Zustand_1	hmm.State	2780950		Zustand_2 (1913643)
3	1913643	Zustand_2	hmm.State		Zustand_1 (20196260)	

Emitter

2780950 (hmm.SimpleEmitter)

#	Typ	Symbol	W'keit
1	org.biojava.bio.symbol.AlphabetManager$WellKnownAtomicSymbol	cytosine	0.0
2	org.biojava.bio.symbol.AlphabetManager$WellKnownAtomicSymbol	thymine	0.0
3	org.biojava.bio.symbol.AlphabetManager$WellKnownAtomicSymbol	guanine	0.3
4	org.biojava.bio.symbol.AlphabetManager$WellKnownAtomicSymbol	adenine	0.7

Transitionen

#	Von	Nach	W'keit
1	20196260	1913643	1.0

Abbildung 3.6: Screenshot der HTML-Darstellung eines HMM. Die Tabellen beziehen sich auf ein HMM mit zwei Zuständen (zusätzlich zum magischen Zustand), einem Emitter und einer Transition. Sie enthalten Informationen über Emissions- und Transitionswahrscheinlichkeiten sowie Vorgänger- und Nachfolgerzustände.

wurden insbesondere Objekte der Klasse `ProfileHMM` betrachtet, die für die Implementierung eines Multiplen Sequenzalignments wie in Abschnitt 3.2 beschrieben verwendet wurden. Bei diesem Problem treten die genannten Schwierigkeiten besonders deutlich und gut beobachtbar auf, da die Anzahl der Transitionen und Zustände eines Profile HMM linear mit der Sequenzlänge wächst. Außerdem lassen sich `ProfileHMM`-Objekte aufgrund ihrer linearen Struktur leicht in beliebiger Größe erstellen und analysieren. Eine Untersuchung des Speicherbedarfs der einzelnen Komponenten des Profile HMM ergab, dass allein die Transitionen 60.8% des benötigten Speichers beanspruchen. Eine Reduktion des Speicherbedarfs von Transitionen war also besonders erfolgversprechend.

In der zunächst benutzten Klasse `HMM`, von der `ProfileHMM` erbt, wurden Transitionen als Objekte einer eigens implementierten Klasse `Transition` erzeugt und in `java.util.HashMap`s gespeichert (siehe Kapitel 3.1.2.1). Es werden zwei Maps verwaltet, von denen eine die Transitionen *von* und eine die Transitionen *zu* einem Zustand speichert. Diese redundante Speicherung vereinfacht und beschleunigt Algorithmen wie den Viterbi-Algorithmus (siehe Abschnitt 2.4.1) deutlich. In diesen Datenstrukturen wird so jedem Zustand eine Menge von Transitionen zugeordnet. Diese Implementierung sichert eine einfache Handhabung von Transitionen sowie einen schnellen Zugriff. Aus Sicht der Speichereffizienz ist dieses Vorgehen jedoch nicht vorteilhaft. Die Verwaltung von Transitionen als Objekte bringt einen zusätzlichen Aufwand mit sich. Außerdem ist die Speicherung in einer `HashMap` nicht effizient. Bei Verwendung dieser Implementierung eines HMM beträgt der Speicherbedarf für ein Profile HMM mit 1000 Spalten, somit 3002 Zuständen und 9009 Transitionen, und einem Emissionsalphabet von vier Zeichen etwa 2719KB.

Eine Verringerung des Speicherbedarfs ließ sich durch eine geänderte Verwaltung der Transitionen erzielen. Eine Zuordnung von Nachfolger- und Vorgängerzuständen zu jedem Zustand blieb aufgrund der Wichtigkeit für den Viterbi-Algorithmus also weiterhin bestehen. Für ein HMM mit $|S|$ nummerierten Zuständen kann diese generell durch zwei Mengen S_N, S_V geschehen. Diese enthalten als Elemente die Mengen S_{N_i} und S_{V_i}, $1 \leq i \leq |S|$, welche die Indizes der Vorgänger- bzw. Nachfolgerzustände enthalten. Um auf die Transitionsobjekte verzichten zu können, müssen die Transitionswahrscheinlichkeiten gespeichert werden. Dies ließ sich durch eine Erweiterung der Elemente der Mengen S_{N_i} erreichen: Anstelle von Indizes enthält S_{N_i} nun 2-Tupel, die nicht nur den Index des Nachfolgerzustands, sondern auch die Transitionswahrscheinlichkeit enthalten. Die Mengen S_N, S_V, S_{N_i} und S_{V_i} wurden als Objekte vom Typ `java.util.Vector` implementiert. `Vector` ist flexibler als beispielsweise ein Array, ohne den Speicherbedarf einer schnelleren Datenstruktur wie einer `HashMap` zu erzeugen. Durch diese Änderungen konnte der Speicherbedarf eines Profile HMM mit 1000 Spalten auf 1941KB gesenkt werden, was relativ zur vorherigen Größe einer Einsparung von 28.6% entspricht.

Eine speichersparende Alternative stellt eine Transitionsmatrix dar, in der nur Transitionswahrscheinlichkeiten gespeichert werden. Die Existenz einer Transition kann mit einem von Null verschiedenen Matrixeintrag gleichgesetzt werden. Dieses Vorgehen ist zwar in einigen Fällen besonders speichersparend, jedoch bringt es einige Probleme mit sich, weshalb diese Alternative nicht umgesetzt wurde. Ein wichtiges Problem ist, dass diese Speicherung für die meisten HMM nur dann effizient ist, wenn besondere Datenstrukturen für dünn besetzte Matrizen benutzt werden. Des Weiteren handelt es sich bei

einem veränderbaren HMM um eine Matrix mit nicht festgesetzten Dimensionen, weil sich beim Hinzufügen und Entfernen von Zuständen die Anzahl von Zeilen und Spalten ändert. Die Implementierung einer Matrix mit den benötigten Eigenschaften wäre einerseits vergleichsweise zeitintensiv und andererseits nicht erfolgversprechender gewesen als die implementierte Lösung.

Neben der Speicherung der Transitionen haben die Zustandsobjekte den zweiten wesentlichen Anteil am Speicherbedarf. Anders als bei den Transitionen kann jedoch nicht auf Objekte zur Repräsentation von Zuständen verzichtet werden. Eine mögliche Vereinfachung ist die Aufhebung der Trennung von Zustand und Emitter. Da `Emitter` ein Interface ist, konnte eine neue Klasse `SingleEmitterState` von `State` abgeleitet werden, die das Interface `Emitter` selbst implementiert. Auf diese Weise kann die Anzahl von Objekten im Speicher um die Anzahl von Zuständen mit einem Emitter reduziert werden. Dieses Vorgehen bringt zwei Einschränkungen mit sich: Ein `SingleEmitterState` hat genau einen Emitter und sein Emissionsalphabet ist unveränderbar. Die zweite Einschränkung ergibt sich aus der Verwendung von Arrays zur Speicherung von Emissionswahrscheinlichkeiten. Um eine allgemeine Begrenzung der Flexibilität zu vermeiden, wird `SingleEmitterState` nicht von der Klasse `ProfileHMM`, sondern nur von der Klasse `SimpleProfileHMM` benutzt. Ein `SimpleProfileHMM`-Objekt mit 1000 Spalten belegt 1612KB Speicher. Das entspricht zusätzlich zur Einsparung durch die Änderung der Verwaltung der Transitionen einer Reduktion des Speicherbedarfs um 12.1% gegenüber dem ursprünglichen Bedarf.

Diese Angaben beziehen sich auf den Speicher, der auf dem Java-Heap belegt ist. In der Realität wird dieser Speicherverbrauch zum Teil deutlich größer sein, weil Java stets mehr Speicher reserviert als aktuell benötigt wird. Die Differenz kann abhängig vom Betriebssystem variieren. Durch die Maßnahmen zur Reduzierung des Speicherbedarfs konnte auch die Menge dieses zusätzlich reservierten Speichers gesenkt werden. Der reale Speicherbedarf für das Profile HMM mit 1000 Spalten in der letzten beschriebenen Variante betrug bei Messungen unter Windows XP SP2 und Linux (Kernel 2.6.11.4) mindestens 1793KB. Somit liegt die real gemessene Reduzierung des Speicherbedarfs bei etwa 47.7%.

Um diese Messungen beurteilen zu können, kann zunächst ein Vergleich mit der Implementierung eines Profile HMM aus dem BioJava-Paket angestellt werden. Unter gleichen Testbedingungen belegt ein mit BioJava erstelltes Profile HMM mit 1000 Spalten 5494KB. Damit ist die Implementierung im `HMM`-Paket deutlich effizienter. Dazu muss bemerkt werden, dass die Konzepte der Pakete sehr verschieden sind und deshalb nur eine eingeschränkte Vergleichbarkeit gegeben ist. Unklar ist noch, wie weit der Speicherbedarf reduziert werden könnte. Dazu soll eine grobe Berechnung des minimalen Speicherbedarfs vorgenommen werden. In jeder Spalte sind drei Zustände enthalten, von denen die meisten eines Profile HMM mit 1000 Spalten ein Namensstring aus fünf Zeichen besitzen (abhängig vom Index). Zur Speicherung der Zustandsnamen werden also etwa $3 \cdot 5 \cdot 4 = 60$ Byte benötigt. Zwei dieser Zustände speichern Emissionswahrscheinlichkeit als `double`-Werte für je vier Emissionsobjekte. Hierfür werden also $2 \cdot 4 \cdot 8 = 64$ Byte benötigt. Schließlich enthält eine Spalte noch neun Transitionen, von denen für jede zumindest die Transitionswahrscheinlichkeit gespeichert werden muss, was hier ebenfalls als `double`-Wert ge-

schieht. Es werden also weitere $9 \cdot 8 = 72$ Byte benötigt. Diese überschlägige Berechnung zeigt, dass theoretisch etwa $60 + 64 + 72 = 196$ Byte Speicher pro Spalte ausreichen. Der gemessene Speicherbedarf ist jedoch deutlich höher. Einen Überblick über die genannten Messwerte gibt Tabelle 3.1. Der Unterschied zwischen den theoretischen und den gemessenen Werten lässt sich damit begründen, dass die Daten nicht als primitive Datentypen, sondern in Objekten gespeichert werden. Dadurch müssen nicht nur die Daten selbst, sondern auch Referenzen auf die die Daten haltenden Objekte gespeichert werden. Zum Teil ist das auch über mehrere Ebenen von Objekten hinweg notwendig. Dadurch entsteht zusätzlicher Speicheraufwand. Die Verwendung von objektorientierter Programmierung ist daher mit einem deutlich bemerkbaren Zusatzaufwand verbunden. Allerdings ist sie auch wichtig für die Flexibilität und Benutzerfreundlichkeit der Implementierung. Auf die weitere Senkung des Speicherbedarfs wurde aufgrund des hohen Aufwands verzichtet und die beschriebene Implementierung als Kompromisslösung beibehalten.

Profile HMM erstellt mit	Speicherbedarf
BioJava	5494 KB
`hmm.ProfileHMM` (erste Version)	2719 KB
`hmm.ProfileHMM` (zweite Version)	1941 KB
`hmm. SimpleProfileHMM`	1611 KB
Schätzung der unteren Grenze	192 KB

Tabelle 3.1: Die Tabelle zeigt den Speicherbedarf eines Profile HMM in verschiedenen Implementierungen. Die Angaben beziehen sich jeweils auf ein HMM mit 1000 Spalten. Gemessen wurde der auf dem Java-Heap belegte Speicher. Die Schätzung der unteren Grenze beruht auf der theoretischen Minimalgröße.

3.1.6 Test der Implementierung

Zur Gewährleistung von zuverlässig und stabil funktionierender Software wurden Tests mithilfe des verbreiteten Pakets `JUnit` [BG98] durchgeführt. Die Implementierung der Tests geschah weitgehend parallel zur Entwicklung des `HMM`-Pakets. Dies ermöglichte, dass bei Änderungen in den Klassen überprüft werden konnte, ob diese neue Fehler hervorgerufen haben. Da es während der Entwicklung noch zahlreiche Änderungen gab, erwies sich dies als sehr hilfreich. Das Paket wurde schon in einer frühen Phase der Implementierung von anderen Gruppen verwendet, wodurch zusätzlich zu den Tests Berichte über Probleme und Fehler im praktischen Einsatz in die Entwicklung mit einfließen konnten. Um die Abdeckung des Codes zu überprüfen wurde das Eclipse-Plugin „Coverlipse“ [Kem05] benutzt. Dieses zeigt auf, welche Programmzeilen bei einem Testdurchlauf benutzt werden. Durch die Verwendung des Tools kann abgesichert werden, dass die erstellten Tests der Klassen `HMM`, `SimpleHMM`, `HMMToolbox` und `ProfileHMM` alle Klassen des Pakets abdecken.

3.2 Multiples Sequenzalignment mit Evolutionären Algorithmen

Eine Aufgabe der Projektgruppe war, ein Multiples Sequenzalignment mit Hilfe von Evolutionären Algorithmen (siehe Kapitel 2.5) zu implementieren. Dabei wurde die Optimierung direkt auf den Symbolsequenzen durchgeführt. Am Anfang der Arbeit stand eine Literaturrecherche über das Alignieren von Sequenzen mit Evolutionären Algorithmen. Dabei wurde eine Arbeit von Leopold [Leo04] gefunden, die sich mit diesem Thema befasst. Das Ziel der Gruppe war es, die ermittelten Ergebnisse dieser Arbeit zu prüfen und zu bestätigen. Eine Hauptaufgabe bestand darin, die in der Arbeit vorgestellten Mutations- und Rekombinationsoperatoren zu implementieren. Des Weiteren musste die Anbindung an das Dortmunder ES-Paket implementiert werden. Zum Schluss bot sich durch die Verwendung des Dortmunder ES-Pakets die Möglichkeit, die Erkenntnisse der Arbeit von Leopold mit Hilfe des ebenfalls an der Dortmunder Universität entwickelten Werkzeugs zur Parameteroptimeriung (siehe Kapitel 2.6.3) zu erweitern.

3.2.1 Eingabe und Ausgabe

3.2.1.1 Einlesen der Eingabedaten

Ein Arbeitspunkt lag darin, die Testdaten, DNA-Sequenzen, vor der Ausführung des EA einzulesen und in einem internen Datenformat abzuspeichern. Dazu musste eine eigene Eingabeverarbeitung geschrieben werden. Den dabei wichtigsten Teil stellte das Überführen einer Sequenzdatei in die einzelnen Symbole der Sequenzen dar. Diese Aufgabe war neben dem Bewerten eines Alignments über eine Substitutionsmatrix und dem Verwenden der Symbole als atomare Bestandteile einer Sequenz der dritte Anbindungspunkt an das BioJava-Paket (siehe Kapitel 2.6.2).

Für diese Aufgabe wurde die Klasse `Init` erstellt. Dabei wird ein vom BioJava-Paket bereitgestellter `SequenceIterator` verwendet, um die einzelnen Testsequenzen bzw. deren Symbole aus der Datei zu lesen. Für die interne Datenstruktur, die nun in allen weiterführenden Prozeduren verwendet wird, wurde ein zweidimensionales Feld von Symbolen gewählt.

3.2.1.2 Schreiben der Ergebnisse

Die Ausgabe der ES ist eine Ergebnisdatei, welche die Ergebnisse beliebig vieler unabhängiger Durchläufe des Algorithmus zusammenfasst. Dabei werden Werte wie zum Beispiel der beste erreichte Zielfunktionswert, die Größen der Populationen, die Anzahl an Iterationen pro Evolutionslauf oder die Anzahl erzeugter Generationen festgehalten. Für die Erzeugung dieser Ergebnisdatei wurde eine Klasse `EsFileWriter` implementiert, die die interne Datenstruktur in ein vom ES-Paket Dortmund (siehe Kapitel 2.6.1) definiertes Ergebnisformat konvertiert.

3.2.2 Datenstrukturen

Das ES-Paket der Universität Dortmund beruht auf der Grundlage der Evolutionsstrategien, bei dem Probleme auf ein Feld von Fließkommazahlen abgebildet werden. Daher mussten für die erweiterte Datenstruktur eines zweidimensionalen Arrays von BioJava-Symbolen sämtliche interne Klassen des ES-Paketes angepasst werden. Dazu gehörten die Klassen `Population` und `Individual.` Von der Klasse `Individual` wurde die Klasse `IndividualMSA` abgeleitet. Diese Klasse enthält die neue interne Datenstruktur und erweitert die geerbten Methoden um die Fähigkeit, auf dieser Datenstruktur zu operieren.

3.2.2.1 IndividualMSA

Initialisierung

Jedes Individuum des ES-Pakets beinhaltet den Objektparameter, die interne Datenstruktur, einen Fitnesswert und eine Menge von Strategieparametern (siehe Kapitel 2.5.1 und 2.5.5). Die implementierte ES verwendet nur einen Strategieparameter, der die Schrittweite σ für den Mutationsoperator enthält. Der Fitnesswert wird von der Fitnessfunktion (siehe Kapitel 3.2.3.4) gesetzt und beschreibt die Güte der erzeugten Lösung.

Eine zentrale Funktion dieser Klasse ist das Initialisieren der Individuen für die erste Generation. Die in der Hilfsklasse `Init` eingelesenen Sequenzen werden symbolweise in das zweidimensionale Array, das den Objektparameter bildet, übertragen. Dabei berechnet sich die Größe des Feldes in einer Dimension als die Anzahl der Sequenzen und in der zweiten Dimension als Länge des Alignments l_{align}, die wie folgt definiert ist:

$$l_{align} := \lfloor l_{max} \cdot 1.5 \rfloor,$$

wobei l_{max} die Länge der längsten Eingabesequenz ist. Um den Mutationsoperatoren das Erreichen möglichst vieler Lösungen zu ermöglichen, wird l_{max} mit einem Faktor von 1.5 multipliziert. Die Wahl des Faktors entspricht der von Leopold [Leo04] gewählten. Die dadurch entstehenden freien Stellen im Array werden mit Gap-Symbolen aufgefüllt.

Im nächsten Schritt werden die für die erste Generation notwendigen Individuen erzeugt und mit den eingelesenen Daten initialisiert. Um eine effektive Optimierung zu gewährleisten, muss eine Manipulation der Objektparameter geschehen, um so die erste Generation möglichst weit über den Suchraum zu verteilen. Dazu wurde eine Methode implementiert, die die zuvor eingefügten Gap-Symbole am Ende jeder Sequenz zufällig über die Sequenzen verteilt, ohne deren ursprüngliche Abfolge zu ändern.

Weitere Methoden

Um die Individuen für Nachfolgegenerationen zu erzeugen, enthält die Klasse einen weiteren Konstruktor zur Erzeugung eines leeren Individuums, dessen Schrittweite gleichverteilt aus einem Intervall, welches in der Parameterdatei definiert ist, gewählt wird. Um dieses Individuum zu komplettieren, muss über einen Methodenaufruf der Objektparameter hinzugefügt werden.

3.2.2.2 Population

Die Klasse `Population` ermöglicht das Erzeugen, die Auswertung, die Manipulation und die Ausgabe einer Population. Zudem ist diese Klasse der Container für die benutzten Individuen. Die Klasse wurde nur minimal erweitert, um den Anforderungen zu genügen. Zum einen wurde der Konstruktor um den Problemtyp `MSAPROBTYPE` erweitert, so dass beim Erzeugen einer Population auch die zum Problemtyp gehörenden Individuen erstellt werden, und zum anderen wurde der Klasse ein weiterer Konstruktor hinzugefügt, der die Startgeneration erzeugt.

3.2.3 Operatoren

3.2.3.1 Mutationsoperatoren

Um die Alignments in der Mutationsphase zu verändern, wurden drei Typen von Operatoren erstellt. Zum ersten Operatortyp, der eine unterschiedliche Anzahl von zusammenhängenden Symbolen innerhalb einer Sequenz verschiebt, gehören

- der `LocalShuffle`-Operator,
- der `BlockPartShuffle`-Operator sowie
- der `BlockShuffle`-Operator.

Alle drei Operatoren beginnen damit, zufällig gleichverteilt eine der Sequenzen des Alignments und darin wiederum zufällig gleichverteilt eine Stelle auszuwählen. Im Folgenden soll nun näher beschrieben werden, wie die drei Operatoren nach dieser Auswahl fortfahren (siehe dazu auch Abbildung 3.7).

LocalShuffle
Der `LocalShuffle`-Operator verschiebt ein Symbol zufällig nach links oder rechts, abhängig davon, ob sich neben dem Symbol Gap-Symbole befinden. Ist die Verschieberichtung bekannt, wird die Anzahl zusammenhängender Gap-Symbole ermittelt und gleichverteilt bestimmt, um wie viele Stellen das Symbol verschoben wird.

BlockPartShuffle
Ein Block sei im Folgenden eine zusammenhängende Gruppe von Symbolen maximaler Länge. Das heisst, dass sich innerhalb dieser Gruppe kein Gap-Symbol befindet, und die linken und rechten Nachbarn des Blocks (sofern sie existieren) Gap-Symbole sind. Aufgabe des `BlockPartShuffle`-Operators ist es, zufällig den linken oder rechten Teil eines Blocks zu verschieben, sofern sich neben dem Block Gap-Symbole befinden. Der linke Teil des Blocks wird durch die Symbole vom Blockstart bis inklusive des Symbols an einer zufällig gleichverteilt ausgewählten Stelle gebildet, der rechte Teil aus den Symbolen von der ausgewählten Stelle bis zum Blockende. Dadurch wird sichergestellt, dass sich die Position des ausgewählten Symbols durch die Anwendung des Operators in jedem Fall verändert. Wie beim `LocalShuffle`-Operator kann der Teilblock abhängig von der Anzahl zusammenhängender Gap-Symbole auch um mehrere Stellen verschoben werden.

BlockShuffle

Der `BlockShuffle`-Operator ist ein Spezialfall des `BlockPartShuffle`-Operators, bei dem der zu verschiebende Teilblock dem ganzen Block entspricht.

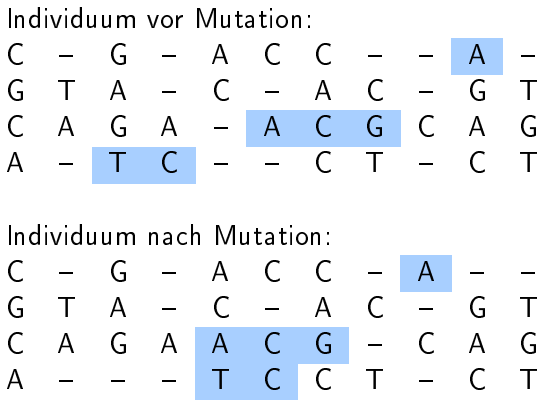

Abbildung 3.7: Die Vorgehensweise aller Shuffle-Operatoren soll anhand des dargestellten Individuums verdeutlicht werden. In der ersten Sequenz wird das Symbol A gewählt und der `LocalShuffle`-Operator angewendet. In der dritten Sequenz wird das Symbol G bei Anwendung des `BlockPartShuffle`-Operators gewählt (hier wird der linke Teilblock verschoben). In der vierten Sequenz wird schließlich das Symbol T gewählt und der `BlockShuffle`-Operator angewendet. Die Ergebnisse der Mutationsprozesse sind dem unteren Individuum zu entnehmen.

ColBuilder

Da sich der Wert eines Alignments spaltenweise berechnet (siehe [Leo04]) und Gap-Symbole dabei immer negativ eingehen, ist es die Aufgabe des zweiten Operatorentyps, in Bereichen großer Ähnlichkeit Spalten ohne Gap-Symbole zu bilden. Dazu sucht der `ColBuilder`-Operator zunächst unter allen Spalten nach denjenigen, die ein „perfect match“ enthalten. Perfect match-Spalten sind jene, die ausschließlich das gleiche Symbol (ausgenommen das Gap-Symbol) enthalten, was vollständige Übereinstimmung der Sequenzen an dieser Stelle bedeutet. Danach werden die perfect match-Spalten bestimmt, aus dessen Vorgänger- oder Nachfolgerspalten gapfreie Spalten erstellt werden können. Nach zufällig gleichverteilter Wahl einer dieser perfect match-Spalten geschieht das Erstellen der neuen gapfreien Spalte schließlich durch Verschieben der entsprechenden Symbole (können sowohl Vor- als auch Nachfolgerspalten gapfreie Spalten werden, wird wiederum zufällig gleichverteilt zwischen den beiden Spalten gewählt). Zur Verdeutlichung der Arbeitsweise soll Abbildung 3.8 dienen.

GapRemover

Der dritte Operatorentyp erkennt und entfernt reine Gap-Spalten in den Alignments. Reine Gap-Spalten müssen entfernt werden, da sie die Fitness verschlechtern, ohne Einfluss auf die Ähnlichkeit der Sequenzen zu haben. Die Methoden zur Erkennung und Löschung

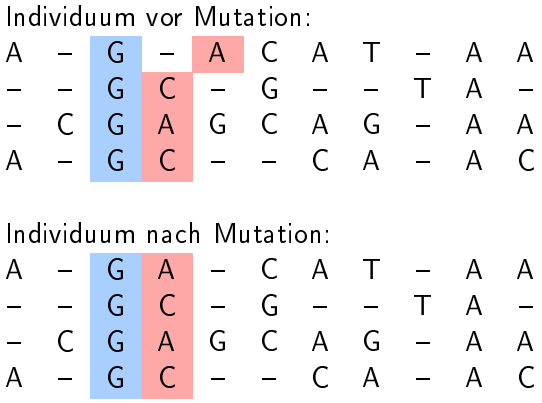

Abbildung 3.8: Funktionsweise des `ColBuilder`-Operators: Die dritte und zehnte Spalte im Individuum besitzen ein perfect match, die blau markierte sei die für die Mutation gewählte. Die Vorgängerspalte kann keine gapfreien Spalte werden, da sich in der zweiten Sequenz links vom G-Symbol nur mehr Gap-Symbole befinden. Die Nachfolgespalte hingegen kann durch Verschieben des A-Symbols in der ersten Sequenz in eine gapfreien Spalte umgeformt werden. Die neue gapfreie Spalte bzw. die entsprechenden Symbole sind in den Individuuen rot markiert.

dieser Spalten werden in der Klasse `MSAGapRemover` bereit gestellt. Im ersten Schritt wird im Individuum nach Gap-Spalten gesucht. Im zweiten Schritt werden diese Spalten zuerst gelöscht und abschließend am rechten Ende des Alignments wieder eingefügt, sofern die Alignmentlänge kleiner als l_{align} (der Länge der Alignments der Startgeneration, siehe Kapitel 3.2.2.1) ist. Dieses Vorgehen wurde gewählt, um zu verhindern, dass dieser Operator über mehrere Generationen sämtliche Gap-Symbole entfernt. Das Entfernen zu vieler Gap-Symbole hätte zur Folge, dass die restlichen Mutationsoperatoren nur sehr stark eingeschränkt neue Lösungen finden könnten, da diese das Vorhandensein von Gap-Symbolen voraussetzen.

3.2.3.2 Rekombinationsoperator

Auch der verwendete Rekombinationsoperator wurde in Anlehnung an den von Leopold [Leo04] vorgestellten Operator implementiert. Dabei werden zwei Lösungen miteinander rekombiniert, indem aus dem ersten Individuum (Elternteil 1) eine gapfreie Spalte in das zweite Individuum (Elternteil 2) übertragen wird.

Das Verfahren startet damit, dass zuerst im Elternteil 1 nach gapfreien Spalten gesucht und danach zufällig gleichverteilt eine davon ausgewählt wird. Wird keine gapfreie Spalte gefunden, wird die Suche im Elternteil 2 fortgesetzt und die beiden Individuen vertauschen ihre Reihenfolge (Elternteil 1 wird zu Elternteil 2 und umgekehrt). Werden wiederum keine gapfreien Spalten gefunden, bildet das Erstellen einer Kopie von Elternteil 1 den Abschluss der Rekombination.

Ansonsten werden im Elternteil 2 jene Symbole gesucht, die im Elternteil 1 die ausgewählte Spalte bilden. Dabei muss die Reihenfolge der Symbole innerhalb der Sequenzen

beachtet werden. Befindet sich beispielsweise das dritte C-Symbol einer Sequenz in der ausgewählten Spalte im Elternteil 1, so muss im Elternteil 2 auch das dritte C-Symbol dieser Sequenz gesucht werden. Sind die Symbole gefunden, wir das neue Individuum wie folgt zusammengesetzt: Alle Symbole, die nicht Gap-Symbole sind und sich links von den gefundenen Symbolen im Elternteil 2 befinden, werden in das neue Individuum kopiert und bilden den linken Teil. Danach wird die ausgewählte Spalte als Ganzes an der ersten freien Position eingefügt. Der rechte Teil des neuen Individuums wird durch das Kopieren aller Symbole, die nicht Gap-Symbol sind und sich rechts von den gefundenen Symbolen im Elternteil 2 befinden, gebildet. Schlussendlich werden alle, durch das Einfügen des linken und rechten Teils entstandenen, Freiräume mit Gap-Symbolen aufgefüllt. Das neue Individuum enthält nun die gapfreie Spalte von Elternteil 1 und die restliche Information von Elternteil 2. Abbildung 3.9 verdeutlicht den Ablauf.

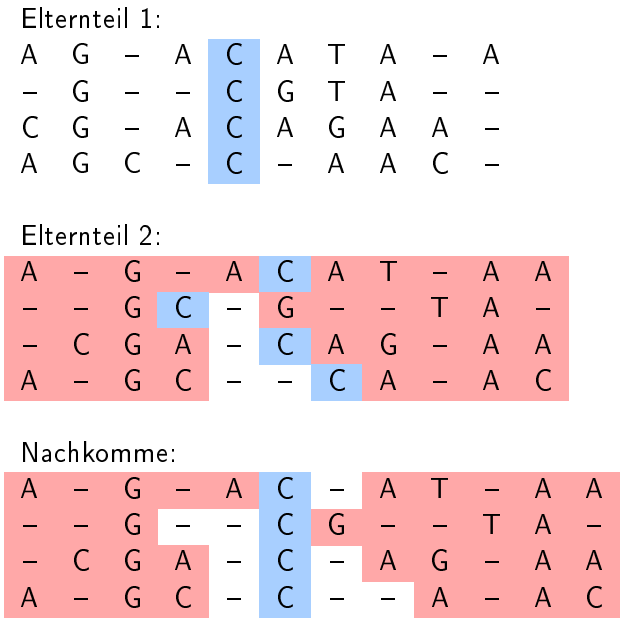

Abbildung 3.9: Die Darstellung schematisiert den Ablauf einer Spaltenrekombination bei Multiplen Sequenzalignments. Dabei wird die blau markierte Spalte des Elternteil 1 dem Nachkommen vererbt. Dazu werden die Symbole der Spalte im Elternteil 2 gesucht und der Nachkomme durch Kopieren des linken und rechten Teils des Elternteil 2 sowie Einfügen der Spalte dazwischen erzeugt.

3.2.3.3 Selektion

In Evolutionären Algorithmen werden zwei Arten von Selektion eingesetzt. Sowohl die Selektion der Eltern (Elternselektion) als auch die Umweltselektion wurden bei der Optimierung von Sequenzalignments benutzt.

Elternselektion

Im Rahmen des Projekts wurde für die Optimierung eines Alignments eine Schnittselektion (truncation selection) als Elternselektion gewählt. Zuerst wird die Elternpopulation sortiert und dann ein Schnitt durchgeführt. Die Hälfte mit den besseren Individuen wird als Pool für mögliche Eltern gewählt. Innerhalb dieser Hälfte werden dann die Eltern zufällig gleichverteilt gewählt. Der Wert, ab dem eine Population geschnitten wird, ist ein einstellbarer Parameter, der problemspezifisch gewählt werden muss. Ist der Wert klein (ca. 10% oder 20%), bedeutet dies, dass nur wenige gute Individuen als Eltern dienen. Durch diese Einstellungen kann eine Population zwar schnell konvergieren, aber auf der anderen Seite kann der Algorithmus in einem lokalen Optimum hängen bleiben. Dem gegenüber bedeutet ein hoher Wert, dass viele zum Teil durchschnittliche Lösungen zur Rekombination zugelassen werden und so eine vielfältige Nachkommenpopulation ermöglicht wird. Der Schnittselektionsparameter hat also direkten Einfluss auf den Selektionsdruck.

Umweltselektion

Es wurde die Umweltselektion aus dem ES-Dortmund-Paket benutzt. Dabei handelt es sich um eine einfache Sortierung aller Individuen der Eltern- und Nachkommenpopulation und einen nachfolgenden Zuschnitt auf die eingestellte Populationsgröße. Dies entspricht einer Plus-Selektion für Evolutionsstrategien (siehe Kapitel 2.5.4), wie sie Beyer und Schwefel in [BS02] beschreiben.

3.2.3.4 Fitnessfunktion

Zur Berechnung des Fitnesswerts einer Lösung wird das Verfahren „Sum-of-Pairs“ verwendet. Bei dieser Methode wird für jede Spalte des Alignments ein Wert über paarweises Vergleichen der Symbole in der Spalte bestimmt und über alle Spalten aufsummiert. Der Spaltenwert ergibt sich als Summe der Kosten aller paarweisen Vergleiche. Der Wert eines Vergleichs ist abhängig von den beiden Symbolen:

- Stehen beide Symbole für Residuen, wird entweder ein Match-Kostenwert vergeben, falls die Symbole gleich sind, oder ansonsten ein Mismatch-Kostenwert.
- Ist eines der Symbole ein Gap-Symbol, so werden Lückeneröffnungskosten vergeben, falls es innerhalb der zugehörigen Sequenz direkt auf ein Residuum folgt. Sollte sich in der Sequenz vor dem Gap-Symbol ein weiteres Gap-Symbol befinden, ergeben sich für den Vergleich Lückenerweiterungskosten, die üblicherweise geringer sind, als Lückeneröffnungskosten.
- Sind beide Symbole Gap-Symbole, erhält der Vergleich keine Bewertung.

Die differenzierte Behandlung von Lücken (siehe dazu auch [DEKM99]) berücksichtigt die Tatsache, dass in Sequenzen öfter Residuenfolgen als einzelne Residuen gelöscht werden. Das bedeutet, dass die Lückenkosten im Wesentlichen das Auftreten einer Lücke bestrafen, während deren Länge geringeren Einfluss auf die Kosten hat.

Die Fitnessfunktion wurde in der Klasse `MSA` implementiert, die von der Schnittstelle `Problem` des ES-Pakets abgeleitet ist. Es handelt sich bei dieser Implementierung um ein Minimierungsproblem, bei dem der Fitnesswert als Distanz der Sequenzen zueinander aufgefasst wird, welche minimiert werden soll. Match- und Mismatch-Kostenwerte werden einer Substitionsmatrix (siehe Abbildung 3.10 und Kapitel 2.3.2.2) entnommen, die mit Hilfe des BioJava-Pakets eingelesen wird.

In Probeläufen wurden verschiedene Werte für Match- und Mismatch-Kosten getestet. Dabei wurden Varianten überprüft, die Mismatches bei den Alignments (siehe Kapitel 2.3) weniger stark bestrafen oder Übereinstimmungen durch besonders geringe Kosten belohnen. Es stellte sich heraus, dass die Bevorzugung von Übereinstimmungen zu besseren Ergebnissen führt. Folglich wurde diese Variante bei den im folgenden Abschnitt durchgeführten Experimenten verwendet.

	A	C	G	T
A	0	3	3	3
C	3	0	3	3
G	3	3	0	3
T	3	3	3	0

Abbildung 3.10: Darstellung einer Nukleotid-Substitutionsmatrix für BioJava. In der ersten Zeile sowie in der ersten Spalte befinden sich die Nukleotid-Symbole. Match-Kostenwerte können der Hauptdiagonalen entnommen werden, die restlichen Matrixeinträge stellen Mismatch-Kostenwerte dar.

3.2.4 Ergebnisse

Experiment: Anwendung einer Evolutionsstrategie zur Berechnung eines Multiplen Sequenzalignments

Vorexperimentelle Planung

In ersten Programmläufen zeigte sich, dass eine erste Version des GapRemover-Operators eine zu stark kontrahierende Wirkung auf das Alignment hat. Schon nach wenigen Generationen war ein Großteil der Gap-Symbole aus dem Alignment gelöscht, was eine starke Beschränkung des Suchraums und dementsprechend schlechte Alignments zur Folge hatte. Eine zweite Version wurde implementiert, die dieses Verhalten, wie im Abschnitt GapRemover beschrieben, verhindert. Weiterhin wurde jedem Mutationsoperator eine durch Versuchsläufe bestimmte Wahrscheinlichkeit zugeordnet, mit der er während der Mutationsphase gewählt werden soll.

Aufgabe

Die Aufgabe bestand darin, die Ergebnisse der Evolutionsstrategie und dessen Operatoren aus der Arbeit von Leopold [Leo04] experimentell zu verifizieren und durch Parameteroptimierung zu verbessern. Des Weiteren wurde ein Vergleich mit einem optimalen

Verfahren zur Bildung von Paarweisen Alignments durchgeführt, um die Güte der durch die Implementierung erzeugten Alignments zu bewerten.

Versuchsaufbau

Es wurden vier Testszenarien mit zehn unabhängigen Läufen durchgeführt. Die Eingabe bestand dabei jeweils aus den ersten 280 Symbolen zweier Nukleotidsequenzen (siehe Tabelle 3.2). Folgende Parameter wurden fest gewählt:

- Matchkosten: 0
- Mismatchkosten: 3
- Gap-Eröffnungskosten: 3
- Gap-Erweiterungskosten: 1
- Funktionsauswertungen: 250.000 pro Lauf
- Der ES-Parameter SLOW (siehe Kapitel 2.6.1.4): 1
- Der ES-Parameter SEED (entspricht dem Startseed für den Zufallszahlengenerator, der in jedem weiteren Lauf um 1 erhöht wird): 120

Der Wert der parametrisierten Parameter NPARENTS, NU, TAU0 und SHI (siehe Kapitel 2.6.1.4) wurde durch das in Kapitel 2.6.3 beschriebene SPOT-Paket für jedes Testszenario bestimmt. Dazu wurden fünf unabhängige Optimierungsläufe mit 30 initialen Designpunkten durchgeführt. Die Parameter wurden in folgenden Intervallen überprüft:

- NPARENTS im Intervall [2,10]
- NU im Intervall [1,5]
- TAU0 im Intervall [1,10]
- SHI im Intervall [1,10]

Resultate und Verifikation

Die ermittelten Ergebnisse des Experiments werden in Tabelle 3.2 Form dargestellt.

Beobachtungen

Die Ergebnisse der Testinstanzen zeigen, dass die Wahl der parametrisierten Werte erheblich von den zu untersuchenden Sequenzen bestimmt wird. Zudem zeigt sich während der Parameteroptimierung, dass die Güte des Alignments stark von den untersuchten Parametern abhängt. In zwei der Szenarien konnte das optimale Alignment gefunden werden.

Sz.	Eingabe	NP.	NU	TAU0	SHI	F.	NW-F.
1	gi|59896243|gb|CY000008.1| gi|63030000|gb|CY001028.1<f|	9	2.5	1.0	3.0	8	8
2	gi|109693428|gb|DQ656119.1| gi|57157354|dbj|AB113659.1|	6	4.5	1.0	2.5	57	45
3	gi|110005864|emb|AM176832.1| gi|110005863|emb|AM176831.1|	6	1.0	2.0	5.0	12	12
4	gi|110005849|emb|AM176817.1| gi|110005848|emb|AM176816.1|	5	5.0	5.2	6.2	130	123

Tabelle 3.2: Darstellung der vier Testszenarien. In der zweiten Spalte befinden sich die Bezeichner der getesteten Sequenzen, die über die NCBI-Datenbank [NCB07] gesucht wurden. Die weiteren vier Spalten geben die durch Parameteroptimierung bestimmten Werte NPARENTS, NU, TAU0 sowie SHI wieder. In den letzten beiden Spalten werden schließlich der Fitnesswert der Implementierung, sowie der Fitnesswert des Optimums (berechnet durch den Needleman-Wunsch-Algorithmus) gegenübergestellt.

Diskussion

Die Ergebnisse des Experiments sind trotz Findens des optimalen Alignments in zwei der vier Szenarien als nicht besonders zufriedenstellend zu beschreiben. In der Großzahl der Läufe wurden nach einer starken Verbesserung des Alignmentwerts im ersten Drittel der durchgeführten Generationen danach vermehrt geringe Verbesserungen erzielt. Hier würde es sich anbieten, das Verhalten der Selbstadaptation der Mutationsstärke genauer zu betrachten. Es könnte sich als vorteilhaft erweisen, die Wahrscheinlichkeit, mit der Mutationsoperatoren in der Mutationsphase gewählt werden, in ähnlicher Weise anzupassen, wie die Mutationsstärke. So wäre es möglich, dass gute Alignments wahrscheinlicher von schwachen Mutationsoperatoren verändert werden als von starken. Zudem wäre eine Verbesserung der Qualität der Alignments durch gezielten Einsatz von biologischem Fachwissen zur Festlegung der Kostenparameter der Fitnessfunktion denkbar.

Ein generelleres Problem bildet die starke Abhängigkeit der Parameter der Evolutionsstrategie von der Eingabe. Es ist nicht möglich, diese Werte einmalig so zu bestimmen, dass sie für verschiedene Eingaben gute Ergebnisse liefert. Die Parameter für jede Eingabe über eine Parameteroptimierung zu eruieren scheint wenig sinnvoll, da die Anzahl der Funktionsauswertungen die des optimalen Needleman-Wunsch-Algorithmus um ein Vielfaches übersteigen würden.

Abschließend wäre festzustellen, dass die implementierte Evolutionsstrategie zwar optimierenden Charakter besitzt, sich aber bezüglich Berechnungszeit und Alignmentgüte - zumindest bei den betrachteten Sequenzen - nicht mit dem Algorithmus von Needleman und Wunsch für Paarweise Globale Alignments messen kann.

3.3 Multiples Sequenzalignment mit Profile HMM

Neben der Erstellung von Multiplen Alignments durch einen Evolutionären Algorithmus unmittelbar auf den Sequenzen (siehe Kapitel 3.2) wurde die Alignierung mittels Profile HMM näher betrachtet. Der klassische Ansatz hierzu sieht vor, ein Profile HMM zunächst auf die gegebenen Sequenzen zu trainieren (vergleiche das Trainingsproblem in Kapitel 2.4). Im Anschluss daran werden die Sequenzen bzgl. ihrer Viterbi-Pfade aligniert (siehe Abschnitt 2.4.3.2). Ein Problem beim Einsatz der klassischen Methoden zum Training ist, dass lokale Extrema während des Trainings nicht überwunden werden und somit nicht zuverlässig, gute Alignments erzeugt werden. Es existieren Ansätze zur Lösung des Problems sowohl im Training (vgl. Krogh et al. [KBSM+94]) als auch bei der Alignierung (vgl. Eddy [Edd95]).

Die in diesem Kapitel beschriebene Methode verwendet zur Vermeidung dieses Problems einen Evolutionären Algorithmus als Basisalgorithmus für das Training. Als Zielfunktion wurde anfänglich die Wahrscheinlichkeit der Viterbi-Pfade genutzt. Dieses Modell wurde später um ein sogenanntes Null-Modell erweitert.

Bei der Implementierung wurden zunächst die in den Abschnitten 2.6.2 und 2.6.1 vorgestellten Pakete BioJava und das ES-Paket der Universität Dortmund benutzt. Die Nutzung von verbreiteter und getesteter Software versprach eine weniger fehleranfällige Implementierung. Während der ersten Implementierungs- und Testphase führten eine für unsere Zwecke unflexible Handhabung der BioJava-HMM und ein damit verbundener Speicheroverhead zu dem Entschluss, eigene HMM samt der benötigten Algorithmen zu implementieren (siehe Kapitel 3.1). Diese wurden zusammen mit der oben genannten Weiterentwicklung der Fitnessfunktion und Verbesserungen in den Mutationen in einer zweiten Implementierungsphase umgesetzt.

Im Folgenden wird zunächst kurz auf die verwendeten Softwarepakete eingegangen. Danach wird die Funktionsweise des Programms sowie die Elemente des EA, also Fitnessfunktion, Mutation und Rekombination näher betrachtet. Am Ende des Kapitels werden Ergebnisse vorgestellt und kritisch reflektiert.

3.3.1 Anbindung an die verwendeten Softwarepakete

Die vorliegende Implementierung hält sich an den durch das ES-Paket Dortmund vorgegebenen Rahmen. Die einzelnen Bestandteile des hier beschriebenen EA sind also Implementierungen der durch das Paket vorgesehenen Schnittstellen. Entsprechend sind die Operatoren Implementierungen der Schnittstellen `IMutation`, `IRecombination` sowie `ISelection`. Des Weiteren wurde, um die Übersichtlichkeit zu erhöhen, die Paketstruktur des ES-Pakets übernommen (siehe Kapitel 2.6.1).

Innerhalb des Pakets musste eine Änderungen an der `ESParameter`-Klasse vorgenommen werden. Da diese keine explizite Übergabe des Problemtyps erlaubt, sondern implizit den paketinternen Problemtyp erwartet, wurde der Pfad von `problem` auf den des eigenen Problemtyps (`hmm.ea.problem`) umgestellt. Aus diesem Grund wird zur Kompilierung des Projekts der Quellcode des ES-Paket Dortmund benötigt.

Die Nutzung des BioJava-Pakets ergab sich aus den zahlreichen im Paket enthalte-

nen Datenstrukturen und Algorithmen zur Manipulation von Sequenzen und Alignments sowie der bereits implementierten Profile HMM (siehe Kapitel 2.4.3 und besonders Abschnitt 2.6.2). Auch nach der Umstellung auf die entwickelten Profile HMM (siehe Kapitel 3.1) ist das BioJava-Paket (Version 1.5, November 2006) für dieses Projekt essenziell.

In der zweiten Projektphase wurden sämtliche Klassen auf das im Rahmen der PG-Arbeit entwickelte eigene HMM-Paket (siehe Kapitel 3.1) umgestellt. Das Paket ermöglicht im Vergleich zu BioJava einen leichteren und damit schnelleren Umgang mit Profile HMM (siehe Kapitel 3.3.4 und 3.3.5). Da der von BioJava verwendete Viterbi-Algorithmus die emissionslosen Zustände nicht korrekt berücksichtigte, musste an dieser für die Laufzeit sehr wichtigen Stelle ein Mehraufwand in Kauf genommen werden. Insgesamt stiegt trotz Verbesserungen an einigen Stellen die Laufzeit nach der Verwendung des eigenen HMM-Pakets sogar.

3.3.2 Funktionsweise des Programms

Das hier zur Sequenzalignierung vorgestellte Verfahren teilt sich in zwei Phasen: In der ersten Phase wird mittels EA ein HMM erzeugt, welches auf die gegebenen Sequenzen trainiert wird, so dass es diese mit möglichst hoher Wahrscheinlichkeit emittiert. Dieses HMM beschreibt eine durch die Eingabesequenzen gegebene Familie an biologischen Organismen. Die Klasse `ESHMMRun` führt dieses Training durch, speichert das erzeugte HMM ab und gibt am Ende ein Alignment der Sequenzen aus.

In der zweiten Phase des Verfahrens wird überprüft, wie ähnlich eine neu gefundene Testsequenz zu der Sequenzfamilie ist, die durch das trainierte HMM beschrieben wird. Hierfür wurde in der zweiten Hälfte des Projekts die Klasse `QuickCompare` implementiert, die als Parameter die selbe Parameterdatei erhält wie das ES-Training sowie das von diesem erzeugte HMM als XML-Datei. Als dritte Eingabe wird die zu vergleichende Sequenz als FASTA-Datei benötigt. Das Programm zeigt dann zunächst das vom ES-berechnete Alignment der Trainingssequenzen an, gefolgt von einem Alignment inklusive der Vergleichssequenz. Schließlich wird eine Bewertung der Alignments durchgeführt und ausgegeben.

3.3.3 Fitnessfunktion

Die untersuchte Fitnessfunktion bewertet ein Profile HMM M bzgl. einer Menge gegebener Sequenzen $\mathbb{X} = X_1, ..., X_m$. Die Bewertung erfolgt bzgl. der Wahrscheinlichkeit der Viterbi-Pfade $V(X_i|M)$ der einzelnen Sequenzen. Analog zur beim Trainingsproblem verwendeten Gesamtwahrscheinlichkeit der Sequenzen bezüglich des Modells (siehe das Trainingsproblem in Kapitel 2.3), werden die Wahrscheinlichkeiten über die Anzahl der Sequenzen multipliziert. Hieraus ergibt sich für die Zielfunktion:

$$f(\mathbb{X}|M) = \prod_{i=1}^{m} V(X_i|M). \tag{3.1}$$

Wie bei der Berechnung der Viterbi-Pfade üblich, wird f nicht direkt berechnet, sondern eine Hilfsfunktion f^* verwendet. Für f^* gilt:

$$\begin{aligned} f^*(\mathbb{X}|M) &= -\log(f(\mathbb{X}|M)) \\ &= -\sum_{i=1}^{m} \log(V(X_i|M)). \end{aligned} \tag{3.2}$$

Durch den Übergang von f zu f^* wird das Gesamtproblem zu einem Minimierungsproblem. Zentraler Berechnungsschritt ist somit die Bestimmung der Viterbi-Pfade zu den Sequenzen. Bei ersten Läufen war eine Tendenz zu übergroßen oder zu kleinen Modellen festzustellen. Diese führten trotz stetig besserer Fitnesswerte nicht zu besseren Alignments. Als Lösungsansatz für diese Problematik wurde im Bereich der Fitnessfunktion die Methode der Null-Modelle geprüft.

3.3.3.1 Null-Modelle

Null-Modelle (engl.: „Null Models“) wurden zuerst im Zusammenhang mit „diskriminierenden Modellen“ (engl.: „Discrimination Models“) beschrieben (vgl. [SGD95]) und beispielsweise von Barrett erweitert (vgl. [BHK97]). HMM werden dort verwendet, um familieneigene von familienfremden Sequenzen zu unterscheiden. Die Bewertung („*Score*“) der Sequenzen erfolgt nicht über das Modell M allein, sondern relativ zu einem Null-Modell ϕ, wobei das Null-Modell Einflüsse modelliert, die man bei der Erkennung der Sequenzen herausfiltern will. Ein einfaches Beispiel ist die Modellierung der durchschnittlichen Verteilung über die Symbole der Gesamtheit aller Protein- bzw DNA-Sequenzen, um vom Modell repräsentierte Sequenzen von zufälligen zu trennen. Die folgende Gleichung macht die Funktionsweise deutlicher:

$$\begin{aligned} \mathit{Score}(\mathbb{X}) &= \log\left(\frac{P(\mathbb{X}|M)}{P(\mathbb{X}|\phi)}\right) \\ &= \sum_{i=1}^{m} \left(\log(P(X_i|M)) - \log(P(X_i|\phi))\right). \end{aligned} \tag{3.3}$$

Diese Art der Bewertung wird „Log-Odds-Scoring“ genannt, da der Quotient aus den beiden Wahrscheinlichkeiten der „Chance “ (engl.: odds) der Sequenz entspricht, von Modell M und nicht vom Null-Modell ϕ erzeugt worden zu sein.

Es lassen sich vier verschiedene Abhängigkeitstypen bzgl. Modell und Sequenzen unterscheiden (s. Barrett et al. [BHK97]):

1. Unabhängig von den betrachteten Sequenzen und Modellen.
2. Fest für alle Sequenzen, aber variabel bezüglich des Vergleichsmodells (Beispiel: Durchschnittsverteilung der Emissionen in den Match-States).
3. Abhängig von den gewählte Sequenzen, aber unabhängig vom Vergleichsmodell (Beispiel: Durchschnittsverteilung der Symbole in den Sequenzen).

4. Abhängigkeit sowohl von Sequenz als auch Modell.

In diesem Projekt wird ein Null-Modell des vierten Typs verwendet. Als Verteilung wird die durchschnittliche Verteilung über die Match-States gewählt. Hierbei wird die Abweichung von jedem Match-State zu dieser Verteilung bewertet. Sequenzen werden somit abhängig von ihrem Viterbi-Pfad belohnt.

3.3.3.2 Umsetzung

Es bezeichne f_ϕ die Fitnessfunktion des Null-Modells:

$$f_\phi(\mathbb{X}|M) = \prod_{i=1}^{m} \frac{V(X_i|M)}{P(\mathbb{X}|\phi)}.$$

Zum Verständnis sei hier nochmal auf die Zusammensetzung des Viterbi-Pfads hingewiesen. Die Wahrscheinlichkeit für den Pfad $p = s_1, .., s_n$ bzgl. einer Sequenz X lässt sich berechnen als Produkt aus den Transitionswahrscheinlichkeiten von s_i nach s_{i+1} mi $i \in \{1, ..., n-1\}$, bezeichnet mit $T(q_{i+1}|q_i) = t_{(i)(i+1)}$, und im Falle eines emittierenden Zustands der Emissionswahrscheinlichkeit des jeweiligen Symbols $P(x|s) = e_{xs}$. Werden die Zustände aufgetrennt in S_m für Match-, S_i für Insert- und S_d für Delete-Zustände, so ergibt sich:

$$V(X|M) = \left(\prod_{k=1}^{n} T(s_{k+1}|s_k)\right) \cdot \left(\prod_{s_m \in S_m} P(x|s_m)\right) \cdot \left(\prod_{s_i \in S_i} P(x|s_i)\right).$$

Für das Null-Modell ergibt sich aus dieser Notation:

$$P(X|\phi) = \prod_{s_m \in S_m} P(s_m|\phi)$$

Substituiert man diese Terme in der Gleichung für f_ϕ, wobei die zur Sequenz X_l gehörigen Zustandspfade mit $p^l = s_1^l \dots s_{n_l}^l$ bezeichnet sind, so ergibt sich:

$$f_\phi(\mathbb{X}|M) = \prod_{l=1}^{m} \left(\prod_{k=1}^{n_l} T(s_{k+1}|s_k) \cdot \prod_{s_m \in S_m} \frac{P(x|s_m)}{P_\phi(x|s_m)} \cdot \prod_{s_i \in S_i} P(x|s_i)\right). \tag{3.4}$$

Geht man wie in 3.3.3 beschrieben von f zu f^* über erhält man also:

$$\begin{aligned} f_\phi^*(\mathbb{X}|M) = \sum_{l=1}^{m} (& \sum_{k=1}^{n_l} log(T(s_{k+1}|s_k)) \\ & + \sum_{s_m \in S_m} (log(P(x|s_m)) - log(P_\phi(x|s_m))) \\ & + \sum_{s_i \in S_i} log(P(x|s_i))). \end{aligned} \tag{3.5}$$

Der Strafterm kann also direkt während des Durchlaufs der Viterbi-Pfade berechnet werden.

3.3.3.3 Implementierung

Während die Funktionalität des BioJava-Pakets die Wahrscheinlichkeiten bei Aufruf der Viterbi-Funktion direkt zur Verfügung stellt, werden nach der Umstellung auf die eigene Implementierung die Viterbi-Pfade durchlaufen und die logarithmischen Wahrscheinlichkeiten wie in Gleichung 3.2 bzw. für das Nullmodell in 3.5 beschrieben aufsummiert. Die Berechnung wird in der Individuenklasse `hmm.ea.population.IndividualProfileHMM` durch den Aufruf der Methode `evaluateInd(EsParameter)` angestoßen. Die eigentliche Berechnung erfolgt in der Problemklasse `hmm.ea.problem.ProfileHMMOpt` in der Methode `evaluate(SimpleProfileHMM)`.

3.3.4 Mutation

Der implementierte EA stellt drei Mutationsoperatoren zur Verfügung, die alle topologischen Merkmale eines Profile HMM erfassen sollen. Dies sind (siehe Kapitel 2.4.3) die Emissions- sowie Transitionswahrscheinlichkeiten und die Länge des Modells, also die Anzahl der Spalten. Die Operatoren sind in der Paketstruktur zu finden als:

- `hmm.ea.operator.ProfileHMMParMutation`
- `hmm.ea.operator.ProfileHMMTransMutation`
- `hmm.ea.operator.ProfileHMMColMutation`

Nachfolgend sollen diese beschrieben werden.

3.3.4.1 Mutation der Emissionswahrscheinlichkeiten

Die Mutation der Emissionswahrscheinlichkeiten verändert in jedem Individuum der Population die Emissionsverteilung eines Zustandes. Dazu wird jeweils zuerst gleichverteilt ausgelost, ob ein Match- oder Insert-State mutiert werden soll. Anschließend wird ebenfalls zufällig gleichverteilt eine der Spalten des HMM gewählt, womit der zu mutierende Zustand eindeutig bestimmt ist. Die Emissionswahrscheinlichkeiten der einzelnen, von diesem Zustand erzeugten Zeichen werden nun unabhängig von ihren bisherigen Werten auf einen Zufallswert gesetzt und anschließend werden die so erzeugten Emissionswahrscheinlichkeiten für die vier Symbole normiert, so dass ihre Summe wieder 1 ergibt.

3.3.4.2 Mutation der Transitionswahrscheinlichkeiten

Zur Mutation der Transitionswahrscheinlichkeiten wird prinzipiell dasselbe Verfahren verwendet. Zunächst wird gleichverteilt ein Match-, Insert- oder Delete-State gewählt und danach die Position im Modell, d.h. die Spalte, bestimmt. Die Änderung an der Verteilung der ausgehenden Transitionen wird dann analog zu den Emissionswahrscheinlichkeiten durchgeführt.

3.3.4.3 Mutationsstärke

Die Mutationen der Transitionen und Emissionen veränderten nach der ersten Implementierungsphase diese Werte immer zufällig und führten während jeder Generation zu einer globalen Suche. Durch die Einführung einer Mutationsstärke wurde dafür gesorgt, dass während der ersten Generationen weiter eine globale Suche durchgeführt wird, um den Suchraum zu sondieren. In späteren Generationen wird die Mutationsstärke (abhängig von der Verbesserung der Fitness) so verändert, dass während der letzten Generationen eine lokale Suche durchgeführt wird, da zu erwarten ist, dass in der Umgebung eines im Suchraum zufällig gefundenen guten Punktes weitere gute, wahrscheinlich sogar bessere Punkte liegen. Eine Suche in der Umgebung dieser guten Punkte ist dann vielversprechender als eine globale Suche. Die implementierte Mutationsstärke ist für die erste Generation 1 und konvergiert während der letzten Generationen gegen 0. Zur Bestimmung der Konvergenzgeschwindigkeit wird nach jeder Generation überprüft, ob in dieser ein Modell mit besserer Fitness gefunden wurde. Die Mutationsstärke wird dann auf

$$Mutationsstaerke = \frac{1}{2}(1 + \cos(\pi \cdot \frac{AktuelleGen. - Gen.ohneFitnessverb.}{Gesamtgen. - Gen.ohneFitnessverb.})) \quad (3.6)$$

gesetzt. Der Graph dieser Funktion verläuft im betrachteten Intervall $[0, 1]$ wie in Abbildung 3.11) dargestellt. Ohne Berücksichtigung der Fitnessverbesserungen der Generationen (d.h. $Gen.ohneFitnessverb.$ wäre stets 0) würden die Konvergenzgeschwindigkeit genau mit der Konvergenz der abgebildeten Cosinusfunktion gegen 0 übereinstimmen. Wird in einer Generation keine Verbesserung der Fitness erzielt, besteht eine Tendenz dazu anzunehmen, dass eine Einschränkung des Suchgebiets auf die Region um den aktuell besten Wert erfolgsversprechender ist als die Suche mit den selben Parametern zu wiederholen. Umgekehrt kann man auf eine weitere Verbesserung der Fitness bei einer Suche mit den selben Parametern hoffen, wenn in der aktuellen Generation eine Verbesserung der Fitness erreicht wurde. Dieser Effekt wird durch die Subtraktion der gezählten Generationen ohne Fitnessverbesserung erzielt.

Die in der zweiten Projektphase implementierte Mutation der Emissionswahrscheinlichkeiten (`hmm.ea.operator.ProfileHMMNewParMut`) setzt die Wahrscheinlichkeit nicht auf einen zufälligen Wert, sondern addiert zum bisherigen Wert der Emission eine (0-1-verteilte) Gauß-Zufallszahl, multipliziert mit der Mutationsstärke.

$$RandomWk_{Symbol\ sym} = RandomGaussian \cdot Mutationsstaerke + alteWk$$
$$\text{mit } sym \in \{A, G, C, T\}$$

Anschließend werden die so für jedes mögliche Emissionssymbol erzeugten Wahrscheinlichkeiten wieder auf 1 normiert:

$$NeueWk_{Symbol\ sym} = \frac{RandomWk_{Symbol\ sym}}{\sum_{\forall Symbols} RandomWK_{Symbol\ sym}}$$
$$\text{mit } sym \in \{A, G, C, T\}$$

Dieselbe Formel wird von der neu implementierten Mutation der Transitionswahrscheinlichkeiten `hmm.ea.operator.ProfileHMMNewTransMut` verwendet.

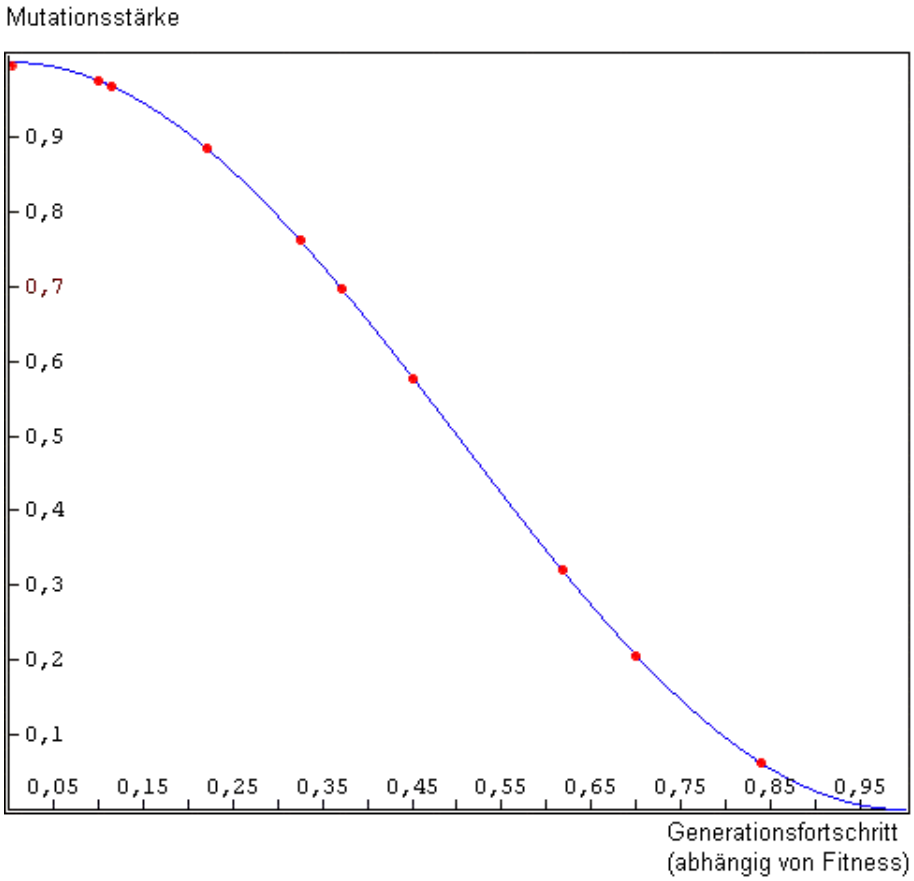

Abbildung 3.11: Mutationsstärke in Abhängigkeit der Generationen bzw. Fitness. Der Generationsfortschritt (x-Achse) ist hierbei eine prozentuale Angabe der jeweiligen Generation modifiziert durch Fitnessverbesserungen (vgl. Formel 3.6). Rote Punkte: Mit dieser Formel berechenete Beispielswerte für 10 Generationen

3.3.4.4 Mutation der Spaltenanzahl

In einem ersten Ansatz wurden in jedem Individuum der Population zufällig gleichverteilt Spalten hinzugefügt oder gelöscht. Die Auswahl der Spalte erfolgte ebenfalls randomisiert nach Gleichverteilung. Eine Ausnahme bildete der Fall, dass das HMM nur aus einer Spalte besteht. Da diese nicht gelöscht werden kann, wurde in diesem Fall immer eingefügt.

Diese Mutation hat, besonders im Vergleich zu den beiden bereits vorgestellten Mutationen, einen sehr starken Einfluss auf das Alignment, da hier in jedem Fall die Viterbi-Pfade der Sequenzen verändert werden. Das rein zufällige Einfügen und Löschen von Spalten führte zu unbefriedigenden Ergebnissen. Aus diesem Grund wurde sie durch eine verbesserte Variante (`ProfileHMMCleverColMut`) ersetzt, die der von Krogh et al. in [KBSM+94] beschriebenen Technik der „Model Surgery“ nachempfunden ist. Die Viterbi-Pfade der Sequenzen werden bei dieser Technik auf eine Häufung von Insert- oder Delete-Zuständen in einzelnen Spalten überprüft. Spalten, deren Delete-Zustände von vielen Viterbi-Pfaden durchlaufen werden und damit dem Überspringen einer Position dienen, werden bei der Auswahl einer zu löschenden Spalten entsprechend bevorzugt gewählt. Hinter Spalten, in denen vermehrt Insert-States durchlaufen werden und die damit nicht alignierende Teilsequenzen repräsentieren, wird mit höherer Wahrscheinlichkeit eingefügt:

Einfügen von Spalten

Es werden zunächst die Viterbi-Pfade der zu alignierenden Sequenzen im aktuellen Profile HMM betrachtet (siehe Abbildung 3.12). Für jeden Insert-State wird die Anzahl

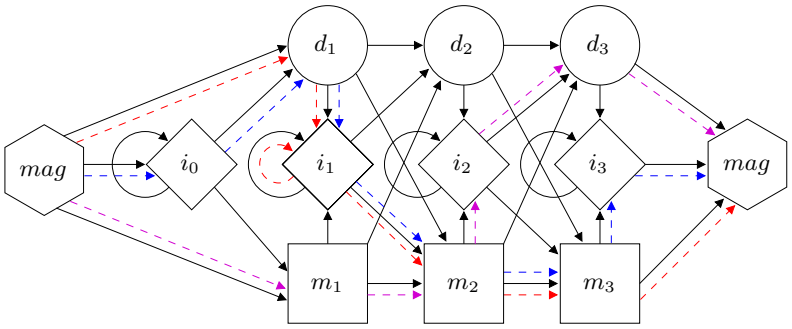

Abbildung 3.12: Mögliche Viterbi-Pfade von 3 Sequenzen

der Besuche durch die Viterbi-Pfade gezählt (siehe Abbildung 3.13). Anschließend wird randomisiert eine Spalte bestimmt, hinter der eine Einfügung günstig sein sollte. Statt einer Gleichverteilung über alle Spalten wie in `hmm.ea.operator.ProfileHMMColMutation` wird als Verteilung die Anzahl der Besuche der Insert-States der einzelnen Spalten benutzt. Die Wahrscheinlichkeit eine Spalte S_i zu wählen, deren Insert-State von allen Viterbi-Pfaden k_{S_i} mal besucht wurde ist, falls von allen Pfaden insgesamt k_{gesamt} Insert-States besucht wurden:

$$P_{insert}(S_i) = \frac{k_{S_i}}{k_{gesamt}} .$$

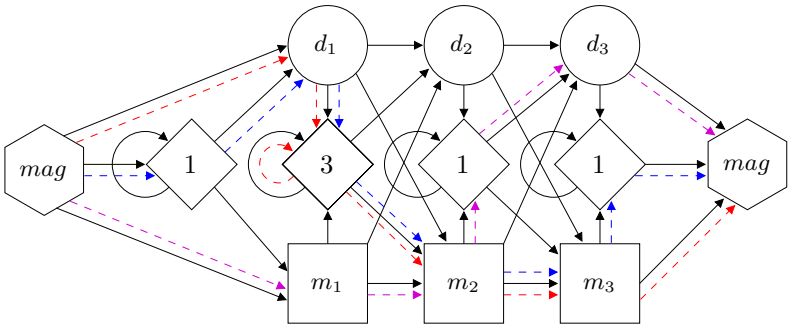

Abbildung 3.13: Gewichtung der Insert-States

Spalten, deren Insert-States sehr häufig besucht werden und damit unähnliche Sequenzabschnitte beschreiben, erhalten also bevorzugt eine Einfügung. Somit besteht die Hoffnung, durch die zusätzliche Spalte an dieser Stelle innerhalb dieses unähnlichen Bereichs einen ähnlichen Teilbereich zu finden. Die Viterbi-Pfade der Sequenzen erhalten in dieser neuen Spalte die Möglichkeit über einen Match-State zu laufen und somit einen Bereich großer Ähnlichkeit anzuzeigen. Sollte kein Viterbi-Pfad einen Insert-State durchlaufen, wird eine Gleichverteilung über alle Spalten für die zufällige Bestimmung der Spalte benutzt.

Löschen von Spalten

Analog zum Einfügen einer Spalte werden hier die besuchten Delete-States gezählt und als Verteilung für die Zufallsauswahl der zu löschenden Spalte benutzt (siehe Abbildung 3.14). Die Wahrscheinlichkeit eine Spalte S_i zu wählen, deren Delete-State von l_{S_i} Viterbi-Pfaden besucht wurde, berechnet sich, falls die Anzahl der besuchten Delete-States summiert über alle Pfade l_{gesamt} ist, entsprechend folgender Formel:

$$P_{delete}(S_i) = \frac{l_{S_i}}{l_{gesamt}} \ .$$

Da Spalten, in denen die Viterbi-Pfade mehrerer Sequenzen Delete-States durchlaufen,

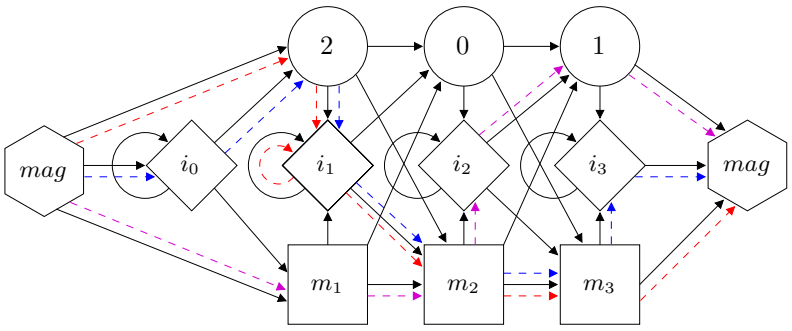

Abbildung 3.14: Gewichtung der Delete-States

mit Spalten des Alignments korrespondieren, in denen viele Sequenzen ein Gap-Symbol enthalten, werden diese Spalten bevorzugt gelöscht. Sollte kein Viterbi-Pfad einen Delete-State durchlaufen, wird wie in `hmm.ea.operator.ProfileHMMColMutation` eine Gleichverteilung über alle Spalten für die zufällige Bestimmung der Spalte benutzt.

3.3.5 Rekombination

Wie in Kapitel 2.5.2 beschrieben, existieren verschiedene Varianten der Rekombination. Aufgrund der Struktur der Profile HMM ist eine intermediäre Rekombination hier nicht adäquat. Eine Berechnung der Mittelwerte von Transitionswahrscheinlichkeiten wäre beispielsweise nur möglich, wenn alle zur Berechnung benutzten Profile HMM die gleiche Spaltenanzahl hätten, wovon nicht ausgegangen werden kann. Stattdessen wird eine diskrete Rekombination benutzt. Bei der implementierten Rekombination werden die Spalten einschließlich ihrer Emissionswahrscheinlichkeiten und ausgehenden Transitionen als atomare Bestandteile eines HMM betrachtet. Innerhalb einer Spalte werden also durch die Rekombination keine Veränderungen vorgenommen.

Wenn ein Nachkomme aus zwei Abschnitten zusammengesetzt werden soll, wird eine reelle Zufallszahl gleichverteilt aus dem Intervall zwischen 0 und 1 erzeugt. Durch Multiplikation mit den Spaltenanzahlen der beiden Eltern und Rundung auf ganze Zahlen werden die Schnittpositionen bestimmt. Der erste Abschnitt des Nachkommens wird nun aus dem ersten Abschnitt des ersten Elternindividuum gebildet, der zweite Abschnitt aus dem zweiten Abschnitt des zweiten Elternindividuums. Es wird angenommen, dass es keine Präferenz für die Reihenfolge gibt, in der die Elternindividuen vorliegen. Diese Form der Rekomination wird auch 1-Punkt-Rekombination genannt (vgl. [Wei02]).

3.3.6 Zusätzliche Parameter der Mutationsoperatoren

3.3.6.1 Mutationsauswahl

Während die Auswahl der drei Mutationsoperatoren in der ersten Phase des Projekts zufällig gleichverteilt erfolgte, sollte mit Hilfe von SPO ein gutes Auswahlverhältnis bestimmt werden. Da sich eine zu häufige Veränderung der Spaltenzahl in Testläufen als ungünstig herausstellte und zudem die Berechnung der Viterbi-Pfade in dieser Mutation sehr zeitaufwändig war, wurden zwei Parameter (Tau0 und Tau1) mit SPO optimiert, die angeben, wie viel mal häufiger eine Transitions- bzw. Emissionsveränderung im Verhältnis zu einer Spaltenmutation stattfinden sollte.

Anzahl an Mutationen pro Generation

Eine wichtige, bisher unbetrachtete Frage ist, wie viele Mutationen innerhalb einer Generation benötigt werden, um zu gewährleisten, dass jede Emissions- und Transitionswahrscheinlichkeit während eines Laufs mindestens einmal mutiert wird. Denn nur bei einer genügend häufigen Veränderung der einzelnen Werte kann davon ausgegangen werden, dass ein genügend großer Teil des Suchraums betrachtet wurde, um ein gutes Ergebnis

zu finden. Zur Bestimmung dieser Zahl wurde folgende Formel verwendet:

$$Anzahl = Länge \cdot (1 - 0.25^{1/(Generationen \cdot 0.75)}).$$

Bei einer gegebenen HMM Länge und einer voreingestellten Anzahl an Generationen kann so die Anzahl an Mutationen bestimmt werden, um nach 75% der Generationen mit einer Wahrscheinlichkeit von 75% jeden Wert mindestens einmal mutiert zu haben. Zur Berechnung wird das äquivalente Ereignis „Es gibt mit höchstens 25% Wahrscheinlichkeit eine Spalte, die nach 75% der Generationen nie verändert wurde“ betrachtet. Zur Approximation dieses Wertes wurde die folgende Formel verwendet, wobei die sich ergebende Ungleichung („höchstens“) durch eine Gleichung ersetzt wurde. Durch einfache mathematische Umformungen erhält man die obige Formel zur Bestimmung der gesuchten Anzahl.

$$(1 - \frac{Anzahl}{Länge})^{Generationen \cdot 0.75} = 0,25$$

Zur Ermittlung der gesuchten, günstigen Anzahl von Mutationen wird eine Variable W (SLOW, siehe Tabelle 3.4) verwendet, die angibt, wie viel mal mehr Mutationen an einer Variable vorgenommen werden sollen, als der oben beschriebene Wert:

$$Anzahl = W \cdot Länge \cdot (1 - 0.25^{1/(Generationen \cdot 0.75)}).$$

Ein Ziel der Anwendung der SPO war es, einen guten Wert für diesen Parameter W zu finden.

3.3.7 Ergebnisse

Experiment: Training von Profile HMM mittels EA zur Bildung von Multiplen Sequenzalignments.

Vorexperimtentelle Planung

Die zunächst verwendeten Profile HMM der BioJava-Klasse stellten sich für die oben beschriebenen Operatoren als unpraktikabel heraus. Durch die starre Struktur der BioJava HMM waren Eingriffe wie das Löschen oder Hinzufügen von Spalten sowie die Rekombination nur durch die Erzeugung neuer HMM mit den gewünschten Parametern möglich. Da in Java die Löschung von Objekten und somit die Freigabe von Speicher nicht explizit möglich ist, wurden schnell Speichergrenzen erreicht. Dies gab Anlass für eine Eigenimplementierung von HMM (siehe Kapitel 3.1).

Ein weiteres Problem dieser ersten Tests war der Einfluss der Fitnessfunktion auf die Länge des Modells. Die Summe der Viterbi-Pfadwahrscheinlichkeiten reichte nicht aus, um eine gute Länge für das Modell zu bestimmen. Dieser Unzulänglichkeit wurde sowohl durch Einführung der im Vorfeld bereits beschriebenen Null-Modelle in der Fitnessfunktion, als auch durch Anpassungen in der Mutation der Spaltenanzahl (siehe Kapitel 3.3.4.4) begegnet.

Nach der Umstellung auf die eigene Implementierung ergaben sich leider neue Schwierigkeiten. Die Umsetzung der HMM litt zwar nicht mehr unter Speicherproblemen, es

ergab sich jedoch eine starke Laufzeitverschlechterung für den Viterbi-Algorithmus. Die einmal pro Sequenzfamilie durchzuführende Trainingsphase dauert je nach Sequenzlänge n mehrere Stunden oder Tage, da die Fitnessberechnung mit der Viterbi-Funktion im wesentlichen eine Laufzeit von $O(n^2 \log n)$ (siehe Kapitel 3.1) hat und in jeder Generation des EA mehrmals aufgerufen werden muss.

Aufgabe

Erstes Ziel war es, die Ergebnisse der Testläufe aus dem 1. Semester zu verbessern und die Funktionsweise des EA auf den HMM genauer zu untersuchen. Zu diesem Zweck wurden die in Kapitel 3.3.6.1 vorgestellten Parameter eingeführt und ihr Einfluss auf die Resultate geprüft.

Versuchsaufbau

Grundlage der folgenden Analyse ist ein SPO-Lauf mit sieben Sequenzen, deren Länge jeweils ca. 20 Zeichen beträgt. Die Sequenzen sind in Abbildung 3.15 dargestellt. Als Fitnessfunktion wurde die in Abschnitt 3.3.3 beschriebene Variante verwendet. Die

```
Sequenz 1: A G T A T G G A A A G A A T A A
Sequenz 2: T A T G G A A A G A A T A A A A
Sequenz 3: A G T A T G G A A A G A A T A A
Sequenz 4: A G T A T G G A A A G A A T A A A A G A
Sequenz 5: A G T A T G G A A A G A A T A A A A
Sequenz 6: A A T A T G G A A A G A A T A A A
Sequenz 7: T A T T C A G C A T G G A A A G A A
```

Abbildung 3.15: Die sieben Sequenzen der SPO-Läufe.

Konfiguration der Parameter kann der Tabelle 3.3 entnommen werden. Zu jeder Parameterkonfiguration wurden drei unabhängige Läufe mit konstanter Zahl an Fitnessfunktionsauswertungen durchgeführt.

Parametername	Kurzbeschreibung	Wertebereich
NPARENTS	Größe d. Elternpopulation	$[1, 10]$
NU	Verhältnis Nachkommen- zur Elternpopulation	$[1, 10]$
TAU0	Anzahl der Transitionsmutation rel. zur Spaltenmutation	$[1, 100]$
TAU1	Anzahl der Emissionsmutation rel. zur Spaltenmutation	$[1, 100]$
SLOW	Mutationsstärke	$[1, 20]$

Tabelle 3.3: Wertebereiche untersuchten Parameter.

Resultate und Visualisierung

Die Tabelle 3.4 zeigt die Resultate der durchgeführten Experimente. Für jeden Parame-

tersatz enthält sie den Mittelwert der erzielten Fitnesswerte. Dieser Mittelwert ist zur Repräsentation ausreichend, da die einzelnen Werte während der Experimente nicht sehr weit um ihr Mittel streuten (die durchschnittliche relative Abweichung liegt bei $0,1$).

Zur Verdeutlichung einiger Schlüsse sind im unterem Teil der Tabelle zudem der Durchschnitt über alle Werte, sowie über die über- sowie unterdurchschnittlichen Läufe angegeben. Da Minimiert wird, wird das Mittel über die unter dem Durchschnitt liegenden Werte als „besser“, die über dem Durchschnitt liegenden mit „schlechter“ bezeichnet.

Beobachtungen

Für die die Populationsgröße bestimmenden Parameter NPARENTS und NU sind keine eindeutigen Aussagen möglich, jedoch brachten größere Populationen oder großes NU während der Experimente keine Vorteile. Dies wird zum einen durch die guten Ergebnisse des (1+1) EA (dritte Zeile der Tabelle) und zum anderen durch die Werte der über- bzw. unterdurchschnittlichen Läufe zumindest angedeutet. Letztere sind für den NPARENTS Parameter $4,00$ für die besseren und $6,00$ für die schlechteren Läufe; bei NU sind es $4,09$ und $5,33$.

Aussagen für SLOW erscheinen anhand der erzielten Resultate nicht möglich. Dazu variieren die einzelnen Werte sowohl bei besseren als auch bei schlechteren Ergebnissen zu stark. Es ist insbesondere kein Einfluss dieses Parameter auf die Güte der Fitnessfunktion feststellbar.

Die stärksten Aussagen lassen sich in Bezug auf die Mutationsauswahl treffen. Hier wird eine Bevorzugung der Mutation der Emissionswahrscheinlichkeiten gegenüber der der Transitionswahrscheinlichkeiten deutlich. Die für diese Wahrscheinlichkeiten maßgeblichen Parameter TAU0 und TAU1 stehen bei den besseren Läufen in einem Verhältnis von $39,80$ zu $67,08$. Bei den schlechteren Läufen kehrt sich das Verhältnis zu $54,30$ zu $35,95$ um. Für die sich daraus ergebenden Wahrscheinlichkeiten TM und PM ergibt sich hieraus ein Verhältnis von $1:2$ bei den besseren Läufen.

Diskussion

Aufgrund der Kürze der Sequenzen sind Erkenntnisse lediglich als Tendenzen zu werten. Die Bevorzugung der Mutation der Emissionen vor der der Transitionen lieferte aber während aller Experimente gute Ergebnisse. Wegen der fehlenden Tendenz für die Mutationsstärke kann der Parameter SLOW in der in Kapitel 3.3.6.1 beschriebenen Form vernachlässigt werden. Der Einfluss eines Parameter für Mutationsstärke im Allgemeinen erscheint aber gerade aufgrund der unklaren Ergebnisse dieser Arbeit als sehr interessant. Desweiteren bietet sich für weitere Untersuchungen aufgrund der guten Ergebnisse mit kleinen Populationen ein (1+1) EA an.

Insgesamt kann festgestellt werden, dass die praktische Frage, ob eine gefundene Sequenz zu einer speziellen Genfamilie gehört, mit diesem Verfahren gelöst werden kann. Zur Erstellung des HMM einer Familie muss zwar sehr viel Vorarbeit durch den EA aufgewendet werden, die Alignmentberechung der Klasse `QuickCompare` erfolgt jedoch in für eine praktische Anwendung akzeptablen Zeit von lediglich einigen Sekunden bzw. Minuten je nach Sequenzlänge.

Fitness	NPARENTS	NU	NOFFSPRING	SLOW	TAU0	TAU1	CM	TM	PM
98,32	5	8,31	42	6,57	19,09	85,47	0,01	0,18	0,81
101,28	2	3,23	6	15,41	6,02	25,42	0,01	0,19	0,78
103,58	1	1,45	1	19,56	14,72	97,07	0,01	0,13	0,86
104,33	3	2,10	6	5,15	6,57	57,52	0,02	0,10	0,88
107,73	3	3,31	10	1,58	74,71	80,64	0,01	0,48	0,52
111,79	2	5,12	10	9,47	44,05	90,77	0,01	0,32	0,67
115,77	4	5,99	24	2,10	99,78	53,31	0,01	0,65	0,35
116,67	8	4,76	38	4,07	2,48	76,15	0,01	0,03	0,96
120,66	5	2,36	12	13,50	81,66	56,36	0,01	0,59	0,41
125,88	7	4,24	30	10,62	48,94	48,13	0,01	0,50	0,49
130,48	9	2,90	26	5,83	24,80	6,02	0,03	0,78	0,19
130,90	7	5,80	41	12,21	66,22	8,00	0,01	0,88	0,11
131,16	6	6,86	41	7,69	32,80	37,71	0,01	0,46	0,53
132,11	1	2,76	3	9,00	58,76	71,22	0,01	0,45	0,54
132,68	9	4,39	40	10,85	31,50	66,83	0,01	0,67	0,32
133,87	6	5,59	34	3,03	69,30	14,11	0,01	0,82	0,17
138,56	5	6,09	30	15,62	86,93	82,51	0,01	0,51	0,48
138,98	7	7,10	50	2,52	53,30	66,22	0,01	0,44	0,55
140,38	6	3,59	22	10,28	6,72	62,98	0,01	0,10	0,89
146,27	7	1,25	9	15,56	29,23	7,58	0,03	0,77	0,20
148,42	3	8,71	26	17,86	94,80	23,90	0,01	0,79	0,20
156,74	3	7,72	23	10,45	80,64	3,07	0,01	0,95	0,04
159,36	9	6,53	59	14,53	70,89	17,25	0,01	0,80	0,19
Durchschnitt:									
127,21	5,13	4,79	25,35	9,71	47,99	49,49	0,01	0,36	0,41
Besser:									
110,60	4,00	4,09	17,90	8,80	39,80	67,08	0,01	0,30	0,60
Schlechter:									
139,99	6,00	5,33	31,08	10,42	54,30	35,95	0,01	0,42	0,27

Tabelle 3.4: Ergebnisse aus dem SPO-Lauf. Die Fitnesswerte für jeweils einen Parametersatz wurden gemittelt. NParents, NU, SLOW, TAU0 sowie TAU1 sind die getesteten Parameter (vgl. Tabelle 3.3). NOFFSPRING ist die Größe der Nachkommenpopulation und ergibt sich aus: $NOFFSPRING = \lfloor NPARENTS \cdot NU \rfloor$. CM, TM und PM geben die Auswahlwahrscheinlichkeiten für die drei Mutationstypen Spalten-, Transitions- und Emissionsmutation an (vgl Abschnitt 3.3.6.1).
Desweiteren sind der Durchschnitt über alle Werte, sowie der Mittelwert über die unterhalb und oberhalb des Durchschnitts liegenden Werte angegeben. Da die Fitness minimiert wird, werden die unter dem Durchschnitt liegenden Werte mit „besser“, die über dem Durchschnitt liegenden Werte als „schlechter“ bezeichnet.

3.4 Genvorhersage

Ein Verfahren zur Genvorhersage ist der komparative Ansatz mit GPHMM (siehe Kapitel 2.4.4), welcher von der Annahme ausgeht, dass kodierende Bereiche während der Evolution speziesübergreifend erhalten bleiben. Mutationen in nicht kodierenden Regionen wirken sich nicht negativ aus und es lassen sich Introns und Exons darüber identifizieren, ob die entsprechenden Abschnitte auf der DNA mit denen anderer Spezies übereinstimmen.

Das verwendete GPHMM enthält zehn Zustände. Diese unterteilen sich in sechs Zustände mit variabler Emissionslänge und vier Zustände mit konstanter Emissionslänge. Die Zustände variabler Emissionslänge repräsentieren Exons, Introns und intergenische Bereiche, wohingegen die Zustände mit fester Emissionslänge Start- oder Endpunkte für variable Abschnitte markieren. Die graphische Darstellung des GPHMM kann Abbildung 3.16 entnommen werden. Der so konstruierte Zustandsgraph unterteilt auf diese Weise die übergebenen Sequenzen in ihre funktionellen Abschnitte. Um diese zu erhalten, muss die wahrscheinlichste Zustandsfolge für die Sequenzen ermittelt werden (siehe Kapitel 2.4.4). Zum Training des GPHMM wird anstatt eines modifizierten Baum-Welch-Algorithmus ein EA verwendet.

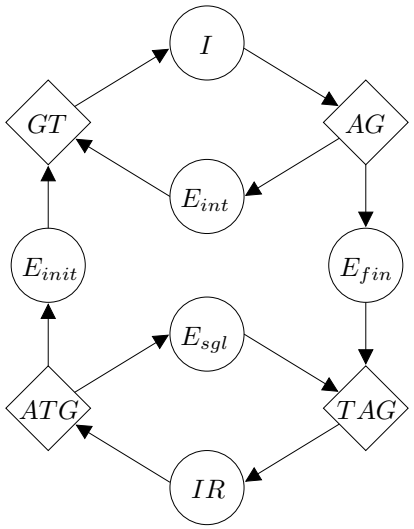

Abbildung 3.16: Das verwendete GPHMM. Die als Rauten dargestellten Zustände (Donor-Splice-Site GT, Acceptor-Splice-Site AG, Startkodon ATG, sowie Stopkodon TAG) haben eine konstante Emissionslänge. Die Kreise stehen für Zustände mit variabler Emissionslänge: Einzelnes Exon E_{sgl}, Startexon E_{init}, inneres Exon E_{int}, Endexon E_{fin}, Intron I und intergenischer Bereich IR.

3.4.1 Eingabedaten

Als Eingabe diente ein Teilergebnis einer Arbeit von Schwartz et al. [SKS+03], in der das komplette menschliche Genom mit einem Maus-Genom aligniert wurde. Zur Durch-

führung dieser Alignierung wurde eine spezielle Version des BLAST-Algorithmus, der BLASTZ-Algorithmus, verwendet. Im Folgenden wird ein kurzer Überblick über die Arbeitsweise von BLAST gegeben.

3.4.1.1 BLAST und BLASTZ

Das von Altschul et al. [AGMM90] entwickelte BLAST (Basic Local Alignment Search Tool) Paket stellt Funktionen [Sou00] zur Verfügung, um Lokale Alignments zwischen einer Anfragesequenz und einer Zieldatenbank zu finden, die jeweils aus DNA- oder Proteinsequenzen bestehen können. Die Idee hinter dem BLAST-Algorithmus ist, dass gute Alignments sehr wahrscheinlich kurze Bereiche mit sehr hoher Übereinstimmung enthalten. Aus diesem Grund werden zuerst solche identischen Übereinstimmungen gesucht und dann als Ausgangspunkte für die Suche nach besseren Alignments verwendet. Diese Suche wird als Erweiterungsprozess bezeichnet, da versucht wird, die gefundenen Übereinstimmungen in beide Richtungen lückenlos zu erweitern. Dabei muss eine Erweiterung den Wert des Alignments vergrößern. Dieser wird zum Beispiel durch Substitutionsmatrizen bestimmt.

BLASTZ von Schwartz et al. [SKS+03] arbeitet in ähnlicher Weise, wobei der Erweiterungsprozess in zwei Teilprozesse zerfällt:

1. Lückenfreie Erweiterung einer gefundenen Übereinstimmung und
2. Jeder dieser gefundenen Übereinstimmungen wird solange mit Hilfe von Lücken vergrößert, bis der Scoring-Wert einen bestimmten Schwellwert unterschreitet.

Ein weiterer Unterschied zu BLAST liegt darin, dass die ausgegebenen Übereinstimmungen (die alignierten Sequenzblöcke) in beiden Eingabesequenzen in gleicher Reihenfolge vorkommen müssen. Die Ausgabe wird durch eine Datei im AXT-Format gebildet.

3.4.1.2 Das AXT-Dateiformat

Eine AXT-Datei setzt sich aus allen Übereinstimmungen des BLASTZ-Algorithmus zusammen, wobei jede davon als Block von drei Zeilen dargestellt wird. Um die Information aus den AXT-Dateien in ein internes Datenmodell zu überführen, wurde die Klasse `AxtParser` erstellt. Ein Block für zwei Sequenzen A und B ist folgendermaßen strukturiert (siehe Abbildung 3.17):

1. Informationszeile, bestehend aus:
   - Treffernummer (beginnend mit 0)
   - Bezeichner der Sequenzen A und B
   - Startindizes des alignierten Bereichs in den Sequenzen A und B
   - Endindizes des alignierten Bereichs in den Sequenzen A und B
   - Richtung, in der auf der Sequenz gesucht wurde (vorwärts (-), rückwärts (+))
   - Alignmentwert
2. Sequenzzeile von Sequenzen A und B

```
0 chr19 3001012 3001075 chr11 70568380 70568443 - 3500
TCAGCTCATAAATCACCTCCTGCCACAAGCCT-GCCTGGTCCCAGGAGACTGCCAC
TCTGTTCATAAACCACCTGCC-TGACAAGCCTGGCCTGTTCCCAAGACACTGCCAC
```

Abbildung 3.17: Darstellung eines Blocks aus einer AXT-Datei. Die erste Zeile enthält wie beschrieben Informationen über den BLAST-Treffer, die darunterliegenden Zeilen bilden das Alignment.

3.4.2 Berechnen des wahrscheinlichsten Pfades

Für GPHMM kann der Viterbi-Algorithmus, der das allgemeine Dekodierungsproblem löst, nicht benutzt werden. Da GPHMM mehrere Zeichen pro Zustand emittieren können, müssen alle wahrscheinlichen Ausgabekombinationen bei der Berechnung der Pfadwahrscheinlichkeiten berücksichtigt werden, was der Viterbi-Algorithmus nicht leisten kann. Zudem können dadurch Zustände im Trellis (siehe dazu 2.4.1.1) übersprungen werden, weshalb der Viterbi keine exakte Lösung liefern kann. Daher wurde ein Verfahren entwickelt, um diese Aufgabe für GPHMM zu lösen.

3.4.2.1 Ermitteln von Signalen

Der erste Schritt bei der Genvorhersage liegt in der Bestimmung von Signalen. Signale sind die biologische Entsprechung von funktionalen Stellen auf einer DNA-Sequenz, die anzeigen, wo ein Gen beginnt, endet oder unterbrochen wird. Diese bilden daher Nukleotidfolgen in einer DNA-Sequenz, die eine bestimmte Länge haben. Außerdem ist die Zuordnung von Signalen und Nukleotidsequenzen nicht eindeutig, sondern es gibt mehrere Realisierungen für ein Signal. Es wurden die folgenden Signale verwendet:

ATG: Kodiert die Aminosäure Methionin und dient in den Genomen fast aller Organismen als Startsignal.

TAG, TAA, TGA: Stehen für das Ende eines Gens und brechen den Transkriptionsvorgang ab. Manche Organismen benutzen nur eine Teilmenge dieser drei Kodons.

GT: Wird als Donor-Splicesite bezeichnet. Die Donor-Splicesite markiert einen Punkt auf der Sequenz, an der ein Intron anfängt und später bei der Erstellung der mRNA an einen anderen Punkt angefügt wird (siehe Kapitel 2.1).

AG: Wird als Acceptor-Splicesite bezeichnet und ist das Gegenstück zur Donor-Splicesite. An dieser Stelle auf der Sequenz wird ein Intron beendet und es fängt ein weiteres Stück eines bereits begonnenen Gens an. Die Donor-Splicesite wird bei der Erstellung der mRNA an die Acceptor-Splicesite gefügt (siehe Kapitel 2.1).

Es gibt noch weitere Signale oder funktionale Bereiche in einer DNA-Sequenz, die untersucht werden könnten. Die vier aufgelisteten beinhalten jedoch die Grundtypen, die untersucht werden müssen, um ein Gen zu finden.

Um die Signale zu ermitteln, wird das eingelesene Alignment nach den vier Signalen durchsucht. Ein Signal wird gefunden, wenn an der selben Stelle der beiden Sequenzen die spezifische Signalfolge gefunden wird. Dabei werden jedoch viel mehr Signale gefunden, als sich tatsächlich in der Sequenz befinden, denn das Startkodon „ATG“ kodiert eine Aminosäure und kann daher auch innerhalb eines Gens vorkommen. Zudem sind die Splicesites „GT“ und „AG“ Bestandteil von vielen Kodons, die Aminosäuren kodieren und daher keine Signale darstellen. Aus diesem Grund muss für diese „Pseudo-Signale“ eine Filterung vorgenommen werden.

Die implementierte Variante der Signalvorhersage sieht die Benutzung von gewichteten Matrizen bei Splicesite-Signalen zur Filterung vor. Dabei wurden statistische Auswertungen über eine Menge von bereits bekannten Genen durchgeführt und die notwendigen Wahrscheinlichkeiten ermittelt. Verschiedene Institute ermitteln diese Wahrscheinlichkeiten, wie zum Beispiel unter [Dir07]. Diese werden genutzt, um einem gefundenen Splicesite-Signal eine bestimmte Vorkommenswahrscheinlichkeit anhand der gewichteten Matrizen zu zuweisen. Wenn die Wahrscheinlichkeit gering ist, wird das Signal verworfen. Auf diese Art werden die echten Signale rausgefiltert. Eine feinere Untersuchung kann mit Hilfe von Markovketten geschehen, die ebenfalls mit Hilfe bekannter Gene trainiert werden. Die Abbildung 3.18 und die Tabelle 3.5 zeigen eine beispielhafte Statistik für ein Donor-Splicesite Signal.

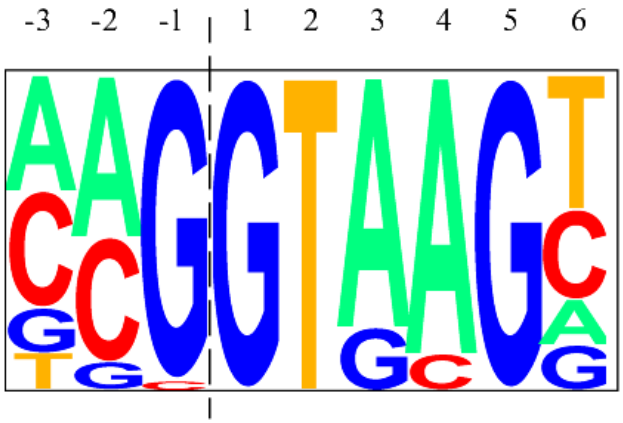

Abbildung 3.18: Bildliche Darstellung einer Gewichtungsmatrix. Die oberste Zeile gibt die Indexpositionen der Residuen an und die Größe des Buchstabens die Höhe der Gewichtung. Stehen in einer Spalte zwei Buchstaben, so sind die Gewichtungen verschiedener Residuen für die jeweilige Position angeben. Die gestrichelte Linie stellt den Fixpunkt der Indizierung dar. Negative Indizies befinden sich im Alignment vor dem betrachteten Signal, und positive enthalten das Signal oder zeigen auf Residuen hinter dem Signal.

3.4.2.2 Erstellen eines Parsegraphen

Ein Parsegraph (siehe Abbildung 3.19) ist ein gerichteter Graph $G = (\mathbb{V}, \mathbb{E})$, dessen Knotenmenge $\mathbb{V}$ den Signalen des zugehörigen Alignments entspricht. Die Kantenmenge

%	-3	-2	-1	1	2	3	4	5	6
A	35	45	0	0	0	80	83	0	15
C	35	38	4	0	0	0	17	0	30
G	15	17	96	100	0	20	0	100	15
T	15	0	0	0	100	0	0	0	40

Tabelle 3.5: Wie Abbildung 3.18 stellt diese Tabelle die Gewichtungswerte für eine Signalsuche dar. Hier wird dies an der Donor-Splicesite „GT“ gezeigt. In der Kopfzeile sind die Indexpositionen der betrachteten Residuen relativ zu einer gedachten Schnittstelle vor der Splicesite dargestellt (Positionen 1 und 2 stehen für „G“ und „T“). Die Werte geben prozentual das Gewicht des jeweiligen Nukleotids an der spezifischen Indexposition an.

für alle $\epsilon \in \mathbb{E}$ ist definiert durch: Für zwei Knoten $v, v' \in \mathbb{V}$ ist $(v, v') \in \mathbb{E}$ genau dann, wenn v' von v aus über genau einen GPHMM Zustand mit variabler Emissionslänge erreichbar ist (siehe Abbildung 3.16). Eine Kante ist somit der (durch den zugehörigen GPHMM-Zustand klassifizierte) Abschnitt im Alignment, der sich zwischen zwei Signalen befindet. Der Parsegraph ist also eine Darstellung aller gültigen Zustandsfolgen für eine

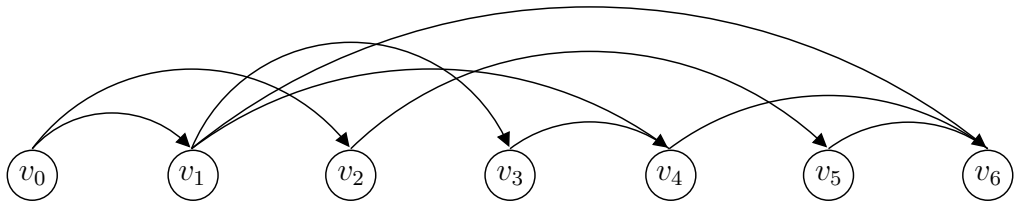

Abbildung 3.19: Darstellung eines Parsegraphen für sieben Signale. Die Knoten repräsentieren Signale im Alignment, die Kanten die Abschnitte im Alignment, die sich zwischen zwei Signalen befinden. Jede Kante typisiert so einen Bereich des Alignments.

Eingabe. Im Folgenden soll beschrieben werden, wie der Parsegraph erzeugt wird.

Der Graph wird mit Hilfe der zuvor bestimmten Liste von Signalen erzeugt. Um die spätere Bewertung des Parsegraphen oder die Pfadsuche zu unterstützen, wurde das Datenmodell so erweitert, dass der Parsegraph einen definierten Startpunkt und Endpunkt besitzt. Alle später gefundenen Pfade sollen mit einem Startkodon beginnen und mit einem Stoppkodon enden. Da in einer gültigen Zustandsfolge (auf Signale reduziert) ein Startkodon immer direkt auf ein Stoppkodon folgt und umgekehrt ein Stoppkodon immer direkter Vorgänger eines Startkodons ist (siehe Abbildung 3.16), werden als erstes Signal ein speziell gekennzeichnetes Stoppkodonsignal, das „virtuelle Startsignal“, und ans Ende des Parsegraphen ein markiertes Startkodonsignal, das „virtuelle Stopsignal“, eingefügt. Diese beiden Signale sind unabhängig von der Eingabe.

Gestartet wird mit dem virtuellen Startsignal, für das alle Startkodonsignale bestimmt werden. Für jedes dieser Startkodonsignale wird eine neue Kante erzeugt, die der Liste ausgehender Kanten des virtuellen Startsignals, der Liste eingehender Kanten der Startkodonsignale, sowie einer Hilfsliste hinzugefügt wird. Diese Hilfsliste beinhaltet Kanten, die im Verlauf der Parsegraphenerstellung noch berücksichtigt werden müssen. Damit ist

die Bearbeitung des virtuellen Startsignals abgeschlossen und die Erstellung des Parsegraphen wird mit dem nächsten Signal in der Signalliste fortgeführt.

Für jedes weitere Signal werden in einem ersten Schritt (siehe Abbildung 3.20) alle Kanten in der Hilfsliste betrachtet. Dies sind Verbindungen, deren Länge noch nicht bestimmt wurde. Die Länge l_ϵ einer Kante $\epsilon = (v, v')$ kann als Anzahl von Residuen zwischen den (durch die Knoten v und v' repräsentierten) Signalen im Alignment aufgefasst werden. Da die Anzahl an Residuen der beiden Teilsequenzen im von ϵ dargestellten Alignmentabschnitt nicht gleich sein muss (Gap-Symbole werden nicht berücksichtigt, da sie keine biologische Relevanz haben), ist l_ϵ wie folgt definiert:

$$l_\epsilon = \lfloor r_\epsilon / 2 \rfloor \,, \tag{3.7}$$

wobei r_ϵ die Anzahl an Residuen im von ϵ repräsentierten Alignmentabschnitt ist. Für jede dieser Kanten wird r_ϵ abschnittweise durch einfaches Zählen der Residuen im Alignmentabschnitt zwischen dem Vorgängersignal in der Signalliste und dem aktuellen Signal bestimmt. Das arithmetische Mittel wird erst dann gebildet, wenn die Kante aus der Hilfsliste entfernt wird.

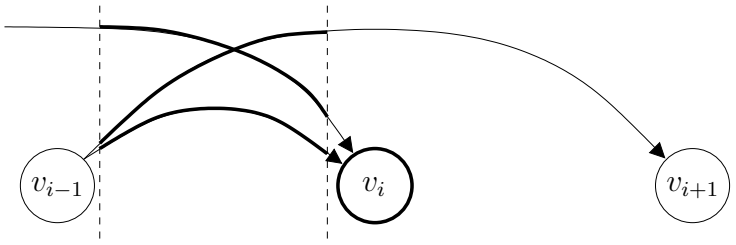

Abbildung 3.20: Im ersten Schritt wird r_ϵ für aller Kanten in der Hilfsliste aktualisiert. Dies geschieht durch Zählen der Residuen im Alignmentabschnitt zwischen dem aktuellen Signal v_i und dessen Vorgängersignal v_{i-1} und dem anschließenden Hinzuaddieren zum alten r_ϵ-Wert.

Im zweiten Schritt (siehe Abbildung 3.21) werden genau die Kanten in der Hilfsliste betrachtet, die im aktuellen Signal enden. Für jede Kante wird gemäß Gleichung 3.7 die Kantenlänge bestimmt und mit einem festgelegten Maximalwert für diese verglichen. Dieser Maximalwert kann sich auf Grund biologischer Erkenntnisse ergeben (falls beispielsweise eine maximale Länge für interne Exons bekannt ist) oder zur Reduktion der Berechnungszeit bei möglichem Informationsverlust (eventuell werden gültige Kanten nicht betrachtet) frei gewählt werden. Überschreitet die Länge den Maximalwert, wird die Kante komplett aus dem Parsegraphen entfernt. Ansonsten wird die Verbindung aus der Hilfsliste entfernt. Im dritten Schritt (siehe Abbildung 3.22) werden für das aktuelle Signal alle möglichen Nachfolgesignale bestimmt, die im GPHMM über einen Zustand mit variabler Emissionslänge erreicht werden können. Danach wird aus dem Positionsabstand zwischen dem aktuellen Signal und einem nachfolgenden Signal (δ_{signal}) eine Mindestlänge $l_{minDist}$ für eine mögliche Kante zwischen diesen beiden Signalen berechnet:

$$l_{minDist} = \lfloor \delta_{signal} / 2 \rfloor \,.$$

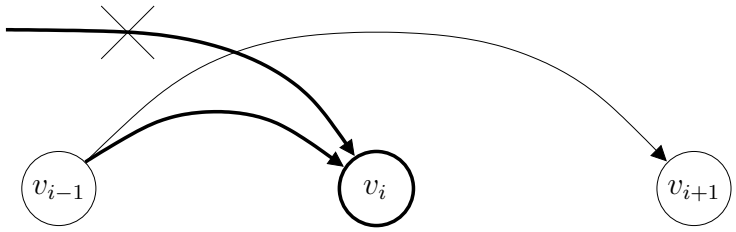

Abbildung 3.21: Im zweiten Schritt werden alle Kanten aus der Hilfsliste, die im aktuellen Signal v_i enden, betrachtet. Überschreitet deren Länge einen festgelegten Maximalwert, so werden sie aus dem Parsegraphen gelöscht (hier sei die obere Kante zu lang), ansonsten ist deren Berechnung abgeschlossen.

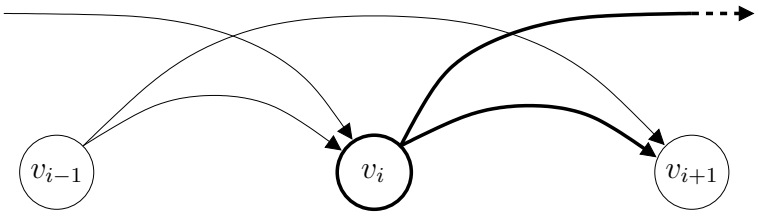

Abbildung 3.22: Im dritten Schritt werden alle ausgehenden Kanten des aktuellen Signals v_i hinzugefügt, sofern deren zuvor berechnete Mindestlänge einen Grenzwert nicht überschreitet.

Die Formel besagt, dass der Abstand der beiden Signale von einander nicht größer ist, als das Mittel der Anzahl der Residuen im Alignmentabschnitt. Diese Abschätzung muss aufgestellt werden, da nicht einzusehen ist, wie viele Lücken in den einzelnen Sequenzen eingefügt wurden. Zur Verdeutlichung der Korrektheit sei auf Abbildung 3.23 verwiesen. Wenn die gesetzten Schranken nicht verletzt werden, wird eine neue Kante erzeugt und der Liste der ausgehenden Kanten des Signals, der Liste eingehender Kanten des Nachfolgersignals und der Hilfsliste hinzugefügt. Andernfalls wird keine Kante erzeugt. Durch die Berücksichtigung der Minimallänge alleine ist jedoch nicht sichergestellt, dass die tatsächliche Länge einer Kante den Maximalwert nicht überschreitet (dies wird im zweiten Schritt überprüft). Allerdings führt sie zu einer wesentlichen Beschleunigung des Grapherzeugungsprozesses.

Ist Schritt drei beendet, ist das aktuelle Signal der Signalliste abgeschlossen und es wird mit dem nächsten Signal fortgefahren. Für das letzte Signal in der Signalliste, dem virtuellen Stopsignal, werden nur noch die Schritte eins und zwei des Verfahrens ausgeführt, wobei die Länge allerdings nicht beschränkt wird. Dadurch ist sichergestellt, dass die wahrscheinlichste Zustandsfolge mit dem virtuellen Startsignal beginnt und dem virtuellen Stopsignal endet. Schließlich sind alle Signale abgearbeitet und der Parsegraph ist erstellt. Für ein besseres Verständnis des Vorgehens soll Algorithmus 3 dienen.

A	T	G	T	–	T	–	T	–	T	A	G
A	T	G	–	T	–	T	–	T	T	A	G

Abbildung 3.23: Die Hälfte des Abstands zwischen den Signalen (grau hinterlegt) beträgt 3. Es ist leicht zu sehen, dass in keiner der Teilsequenzen zwischen den Signalen ein Nukleotid weniger vorkommen darf, da ansonsten zwei Gap-Symbole miteinander aligniert werden, was ausgeschlossen ist. Dadurch ist 3 gleichzeitig die Mindestlänge einer Kante zwischen diesen beiden Signalen.

3.4.2.3 Bewerten eines Parsegraphen

Um den Parsegraphen für die anschließende Pfadsuche vorzubereiten, müssen dessen Kanten bewertet werden. Der Wert $p(\epsilon)$ einer Kante ϵ ergibt sich als Produkt aus Emissions- und Verweildauerwahrscheinlichkeit der Kante:

$$p(\epsilon) = e_\epsilon \cdot d_\epsilon.$$

Die Bewertung des Parsegraphen wird in der Klasse `TransitionScorer` vorgenommen und besteht im Wesentlichen aus einem Durchlauf durch die Signale und deren ausgehende Kanten.

Beginnend beim ersten Signal werden alle ausgehenden Kanten des Signals einer Liste unbewerteter Kanten hinzugefügt. Anschließend wird für alle sechs möglichen Kantentypen (die vier Exontypen, sowie Intron und intergenischer Bereich) die Emissionswahrscheinlichkeit zwischen den beiden Signalen berechnet. Dazu wird der Alignmentabschnitt zwischen den Signalen betrachtet und die Emissionswahrscheinlichkeiten werden entsprechend der Emissionstabellen in den GPHMM-Zuständen, die die Kantentypen repräsentieren, für alle Residuenpaare multipliziert. Danach werden diese Wahrscheinlichkeiten allen unbewerteten Kanten typabhängig aufmultipliziert. Um die Bewertung zu vervollständigen, wird abschließend die Berechnung der Verweildauerwahrscheinlichkeit durchgeführt und zwar immer dann, wenn ein Signal bearbeitet wird, welches eingehende Kanten besitzt. Für alle eingehenden Kanten wird in den zugehörigen GPHMM-Zuständen die Verweildauerwahrscheinlichkeitsfunktion mit der Länge der Kanten aufgerufen und deren Rückgabewert mit dem Wert der Kante multipliziert. Damit ist eine Kante bewertet und wird von der Liste unbewerteter Kanten entfernt. Dieses Vorgehen wird iterativ bis zum letzten Signal durchgeführt. Danach ist der Parsegraph vollständig bewertet. Zur Verdeutlichung der Arbeitsweise soll der Algorithmus 4 dienen.

3.4.2.4 Maximale Pfadsuche

Nachdem der Parsegraph mit allen Transitionen bewertet wurde, muss ein Pfad durch den Parsegraphen gesucht werden, der vom virtuellen Startsignal zum virtuellen Stopsignal führt. Ein Pfad beschreibt eine Abfolge von Signalen, die durch Transitionen miteinander verbunden sind. Die korrekte Abfolge wird durch den Zustandsgraphen sichergestellt, dessen Zustandsverknüpfungen dem biologischen Vorbild nachgestellt sind. Ein Pfad enthält ein Gen, wenn sich ein Startsignal und ein Stopsignal im Pfad befinden und das Stopsignal hinter dem Startsignal liegt.

```
for alle Signale do
    if aktuelles Signal ist virtuelles Startsignal then
        Erzeuge Kanten zu allen Startkodonsignalen;
        Füge Kanten der Hilfsliste hinzu;
        Gehe zu nächstem Signal;
    end
    // Schritt 1
    for alle Kanten aus Hilfsliste do
        Füge Anzahl an Residuen im Bereich Vorgängersignal bis aktuelles Signal zur
        Länge der Kante hinzu;
    end
    // Schritt 2
    for alle im Signal endenden Kanten do
        Entferne Kante aus Parsegraphen, falls sie zu lang ist und kein virtuelles
        Signal enthält;
        Entferne Kante aus Hilfsliste;
    end
    // Schritt 3
    for alle gültigen Nachfolgersignale do
        Falls Minimalabstand zwischen Signalen Maximallänge nicht überschreitet,
        erzeuge neue Kante;
    end
end
```

Algorithmus 3 : Algorithmus zur Erstellung des Parsegraphen. Der Aufbau erfolgt iterativ über alle Signale der Signalliste. Im ersten Abschnitt werden alle ausgehenden Kanten des virtuellen Startsignals erzeugt. In Schritt 1 werden die Längen aller Kanten aus der Hilfsliste stückweise aktualisiert. In Schritt 2 werden Kanten abgeschlossen oder sie werden entfernt, falls sie zu lang sind. Im dritten Schritt werden schließlich alle Kanten zum Parsegraphen hinzugefügt, deren Mindestlänge einen Grenzwert nicht überschreitet.

Diese Abfolge kann mehrfach in einem Pfad vorkommen, daher kann es mehrere Gene pro Pfad geben. Das virtuelle Startsignal ist mit allen sich im Parsegraph befindenden Startsignalen verbunden. Das heißt, dass sich das virtuelle Startsignal in einem intergenischen Bereich befindet und darauf nur der Anfang eines neuen Gens folgen kann.

Um das Dekodierungsproblem (siehe Kapitel 2.4) mit den so aufbereiteten Daten abzuschließen, wird über alle möglichen Pfade, die durch einen Parsegraphen führen, der Pfad gesucht, der die höchste Wahrscheinlichkeit aller Pfade besitzt. Dazu wurden zwei Strategien implementiert.

Vollständige Enumeration

Die vollständige Enumeration iteriert über alle möglichen Pfade. Jedes Startsignal im Parsegraphen entspricht der Wurzel eines Teilbaumes, der durch den Algorithmus 5 tra-

```
for alle Signale do
    for alle GPHMM-Zustände ungleich Signalzustand do
        Emissionswert = 1;
        for alle Residuenpaare zwischen den Signalen do
            Emissionswert = Emissionswert · Wahrscheinlichkeit, dass der
            GPHMM-Zustand das Paar emittiert;
        end
        Speichere Emissionswert;
    end
    for alle unbewerteten Kanten do
        Erhöhe Kantenwert typabhängigen Emissionswert;
    end
    if Signal hat eingehende Kanten then
        for alle eingehenden Kanten do
            Erhöhe Kantenwert um typabhängige Verweildauerwahrscheinlichkeit;
            Entferne Kante aus Liste unbewerteter Kanten;
        end
    end
end
```

Algorithmus 4 : Algorithmus zur Bewertung des Parsegraphen. Die Bewertung erfolgt iterativ über alle Signale der Signalliste. Im ersten Abschnitt werden alle Emissionswahrscheinlichkeiten für das entsprechende Alignment berechnet, um im zweiten Abschnitt typabhängig den Kanten hinzugefügt zu werden. Ein Signal ist abgearbeitet, nachdem alle eingehenden Kanten mit deren Verweildauerwahrscheinlichkeiten berechnet worden sind. Sind alle Signale abgearbeitet, ist der Parsegraph bewertet.

versiert wird. Da der intergenische Bereich vor dem untersuchten Startsignal unterschiedlich groß gegenüber den anderen Startsignalen ist, muss das Durchlaufen der Bäume für jedes Startsignal einzeln durchgeführt werden. Dadurch werden Transitionen teilweise mehrfach bewertet, da sie in mehreren Teilbäumen durchlaufen werden müssen. Durch die komplette Enumeration aller möglichen korrekten Pfade entsteht bereits bei moderaten Problemgrößen eine Laufzeit, die diesen Algorithmus unpraktikabel macht. Um die Laufzeit zu verkürzen wurden einige Parameter hinzugefügt, die bestimmte Annahmen über das Problem berücksichtigen.

So wurde zunächst eine Methode implementiert, die Pfade einer vorgegebenen maximalen Länge verbietet. Das bedeutet, dass bestimmte Äste des Parsegraphenbaumes nicht mehr durchlaufen werden. Der Hintergrund ist, dass bei der Suche nach einem Gen auf einem teilweise bekannten Sequenzstück unnötige Vergleichsoperationen gespart werden können.

Des Weiteren wurde eine Funktion implementiert, die bewirkt, dass Pfade, die eine vorgegebene Wahrscheinlichkeit unterschreiten, ebenfalls beim Traversieren ignoriert werden. Wie bei der maximalen Pfadlänge werden dadurch unnötige Untersuchungen von Teilbäumen unterbunden. Hierbei wird angenommen, dass Pfade, die besonders schlechte

```
for alle Startsignale do
    Erzeuge Kante von virtuellem Start bis zum Startsignal;
    while Startsignal nicht untersuchte Kinder hat do
        Erzeuge einen Pfad bis zu einem Knoten ohne Kinder;
    end
    if erzeugter Pfad korrekt then
        Bewerte Pfad und vergleiche mit bestem bekannten Pfad;
    end
    Gehe Pfad zurück bis Knoten mit noch zu untersuchenden Kindern gefunden
    wird;
end
```

Algorithmus 5 : Der Algorithmus stellt schematisch den Ablauf einer vollständigen Enumeration durch den Parsegraphen dar, um den kompletten Pfad mit der höchsten Wahrscheinlichkeit zu finden.

Ausgangswerte besitzen, sich im weiteren Verlauf nicht mehr stark verbessern. Jedoch ist auch dieser Wert schwer für unbekannte Problemstellungen einzustellen.

Greedy-Strategie

Die zweite Methode für eine Suche nach einem maximalen Pfad (siehe Algorithmus 6) durch einen Parsegraphen arbeitet mit einem Greedy-Ansatz. Anstatt ein globales Optimum finden zu müssen, wird diese Forderung aufgegeben und es werden auch lokale Optima akzeptiert. Der Versuch besteht nun darin möglichst gute lokale Optima zu finden, indem so viel Problemwissen wie möglich im Greedy-Algorithmus eingebettet wird.

```
for alle Signale do
    for alle eingehenden Transitionen des Signals do
        Berechne lokale Pfadwahrscheinlichkeit;
    end
    Schreibe lokale Pfadwahrscheinlichkeit und Vorgänger in Signal;
end
// Backtrace:
Starte mit virtuellem Stopsignal;
while aktuelles Signal ungleich virtuelles Startsignal do
    Setze gemerktes maximales Vorgängersignal vom aktuellen Signal als neues
    Signal;
end
```

Algorithmus 6 : Es wird ein Ablauf einer Strategie skizziert, die durch lokale Entscheidungen an den Signalen einen Pfad maximaler Wahrscheinlichkeit approximiert.

Dabei wird das Konzept der lokalen Pfadwahrscheinlichkeit (LPW) des Viterbi-Algo-

rithmus (siehe Kapitel 2.4.1) benutzt. Hierbei wird einem Signal die Information übertragen, von welcher eingehenden Transition es am meisten Verbesserung der Pfadwahrscheinlichkeit erwarten kann. In einem zweiten Schritt wird eine Rückverfolgung durchgeführt, welche mit dem virtuellen Stopsignal startet und die in den Signalen gespeicherten Verweise auf die wahrscheinlichsten Vorgänger benutzt. Die lokale Pfadwahrscheinlichkeit für ein Signal i berechnet sich folgendermaßen:

$$LPW_i = \max_{\epsilon \in \mathbb{E}} \{LPW_{i-1} \cdot p(\epsilon) \cdot t_\epsilon\}.$$

Dabei ist LPW_i die lokale Pfadwahrscheinlichkeit des Signals i, $p(\epsilon)$ die berechnete Kantenwahrscheinlichkeit aus der Parsegraphenbewertung und t_ϵ die Transitionswahrscheinlichkeit der Kante ϵ.

Somit muss für jedes Signal, wie beim Viterbi-Algorithmus, eine lokale Pfadwahrscheinlichkeit und ein Vorgängersignal abgespeichert werden. Das Verfahren benötigt lineare Rechenzeit. Aus den Tests und den Vergleichen des Greedy-Ansatzes mit der vollständigen Enumeration wird ersichtlich, dass der Greedy-Ansatz zu eingeschränkt ist, um das globale Optimum gut zu approximieren. Eine Erweiterung, die allerdings nicht durchgeführt wurde, besteht in der Ausweitung der Lokalität. Anstatt der ausschließlichen Berücksichtigung des Vorgängers könnte der Baum, der k Vorgänger-Pfade berücksichtigt werden.

3.4.3 Trainieren eines HMM zur Genvorhersage

Für normale Hidden Markov Modelle wird zur Lösung des Trainingsproblems der Baum-Welch-Algorithmus (siehe Kapitel 2.4.2) benutzt. Für GPHMM kann dieser Algorithmus allerdings nicht verwendet werden, da zusätzlich zu den Emissions- und den Transitionswahrscheinlichkeiten auch die Verweildauerwahrscheinlichkeiten trainiert werden müssen.

Der Baum-Welch-Algorithmus setzt die jeweiligen Wahrscheinlichkeiten für ein HMM so, dass eine gegebene Sequenz von dem HMM emittiert würde. Das Problem lässt sich als Optimierungsaufgabe beschreiben, bei der der Abstand einer Emission eines HMM zu einer bestimmten Sollausgabe minimiert wird. Im Rahmen dieser PG wurden Evolutionäre Algorithmen zur Lösung dieser Optimierungsaufgabe gewählt. Für eine erste Analyse der Funktionsweise von Evolutionären Algorithmen zur Lösung des Trainingsproblems wurde eine (1+1)-Evolutionsstrategie benutzt.

Die für Evolutionäre Algorithmen nötige Fitnessfunktion berechnet mit Hilfe eines der Dekodierungsalgorithmen des vorherigen Abschnitts eine Pfadwahrscheinlichkeit, die als Grundlage des Fitnesswertes verwendet wird. Dieser soll durch Variationen am GPHMM maximiert werden. Allerdings sind auch andere Varianten einer Fitnessfunktion möglich, wie zum Beispiel die Berechnung des Abstands eines berechneten Pfades von einem bereits bekannten, optimalen Pfads. Der Vorteil der Maximierung der Pfadwahrscheinlichkeit liegt in der Unabhängigkeit der Methodik von bereits bekannten Daten. Dagegen ist sie nicht zielgerichtet und damit nicht gut auf das Problem angepasst.

Als Variationsoperatoren wurden eine Emissions- und eine Transitionsmutation implementiert. Die Emissionsmutation wählt in einem gegebenen GPHMM zufällig gleichverteilt einen emittierenden Zustand und mutiert eine der Emissionswahrscheinlichkeiten

durch Addition einer gleichverteilten Zufallszahl zwischen 0 und 1. Es wurde eine Schrittweite implementiert, die steuert, wie groß dieser addierte Wert maximal sein darf. Für einen Objektparameter a_i berechnet sich der neue Wert a_{i+1} mit Hilfe einer Gleichverteilung U_Ω und einer Schrittweite δ folgendermaßen:

$$a_{i+1} = \delta \cdot U_\Omega \cdot a_i.$$

Bei der Transitionsmutation wird ebenfalls für einen zufälligen gleichverteilten Zustand eines GPHMM eine beliebige abgehende Transition b_i durch Addition einer gleichverteilten Zufallszahl eine Veränderung bewirkt. Die folgende Formel veranschaulicht dies:

$$b_{i+1} = \delta \cdot U_\Omega \cdot b_i.$$

Beide Mutationsoperatoren können pro Iteration mehrfach ausgeführt werden, was wiederum eine Schrittweite darstellt.

Zudem muss nach jedem Mutationsschritt eine abschließende Normierung durchgeführt werden, damit die Summe aller relevanten Wahrscheinlichkeiten wieder 1 ergibt, was für HMM verlangt wird. Bei der Emissionsmutation wird über alle Emissionssymbole des gewählten Zustandes normiert, bei der Transitionsmutation müssen sich die abgehenden Transitionen zu 1 addieren. Bei der Art und Weise der Mutationsoperatoren werden die Wahrscheinlichkeiten nie subtrahiert, was jedoch durch die Normierung erfolgen kann. Eine Mutation in einem Objektparameter bedeutet für alle zugehörigen Objektparameter eine Anpassung, bei der Wahrscheinlichkeiten mit einem höheren Wert verringert werden.

3.4.4 Verifikation

Die Ausgabe des Programms enthält eine Textdatei der vorhergesagten Gene. Sie unterteilt sich in zwei Abschnitte. Der erste Abschnitt enthält Informationen über die Genpositionen und die Klassifizierung in Intron- und Exonbereiche. Die zweite Hälfte enthält die entsprechend gespleißten Bereiche (siehe Abschnitt 2.1) ohne die Introns für die Verifikation. Verglichen werden diese Daten mit bereits bekannter mRNA aus der NCBI Datenbank [NCB07] über den „Nucleotide-Nucleotide Blast“-Algorithmus. Die Ausgabe des Algorithmus ist eine prozentuelle Übereinstimmung und ein exaktes Alignment mit der übergebenen Anfragesequenz. Stimmt die Anfragesequenz mit einer bekannten mRNA überein, so ist das durch sie repräsentierte Gen richtig vorhergesagt.

3.4.5 Ergebnisse

Experiment: Durchführen eines ES-Trainings für GPHMM zur Genvorhersage

Vorexperimentelle Planung

Erste Experimente zeigten, dass die lange Rechenzeit der vollständigen Enumeration zur Pfadsuche selbst bei Eingaben geringer Länge einen Versuch mit sinnvollen Testdaten unmöglich macht. Aus diesem Grund wurde die in Kapitel 3.4.2.4 eingeführte Greedy-Strategie verwendet. Die Signalfindung wurde mit Gewichtungsmatrizen für Splicesite-Signale durchgeführt. Zur Reduktion der Signalanzahl wurden Splicesite-Signale, die mit

geringeren Vorkommenswahrscheinlichkeiten (siehe Kapitel 3.4.2.1) als −14 (logarithmierter Wahrscheinlichkeitswert) als echte Signale klassifiziert wurden, verworfen. Zur Beschleunigung der Parsegraphenerstellung (siehe Kapitel 3.4.2.2) wurde ein Maximalabstand für Signale von 1000 Residuenpaaren festgelegt.

Um eine Aussage über die Optimierungsfähigkeiten der ES machen zu können, muss im vorhinein bestimmt werden, mit welcher Lösung die Optimierung beginnt. Dazu wurden in dem beschriebenen GPHMM manuell Emissions- und Transitionswahrscheinlichkeiten gesetzt und dieses bewertet. Die Transitionswahrscheinlichkeiten orientierten sich an dem Zustandsgraphen aus Kapitel 3.4, wobei eine Gleichverteilung zu Grunde gelegt wurde. Für die Emissionswahrscheinlichkeiten wurde ebenfalls eine Gleichverteilung über alle Emissionssymbole eines Zustands verwendet. Da die Eingabe aligniert ist und die Annahme davon ausgeht, dass kodierende Bereiche in beiden Sequenzen annähernd übereinstimmen, wurden Symbole, die Übereinstimmung in beiden Sequenzen darstellen, mit einer höheren Wahrscheinlichkeit belegt. Dazu wird für eine Übereinstimmung zweier Symbole ein Parameter benutzt, der für alle Übereinstimmungen zweier Symbole diesen Parameter benutzt. Die restlichen Werte werden gleichmäßig aufgefüllt. Tabelle 3.6 veranschaulicht eine beispielhafte Emissionsmatrix für den Parameter 0.2.

	A	C	G	G	GAP
A	0.2	0.009	0.009	0.009	0.009
C	0.009	0.2	0.009	0.009	0.009
G	0.009	0.009	0.2	0.009	0.009
T	0.009	0.009	0.009	0.2	0.009
GAP	0.009	0.009	0.009	0.009	

Tabelle 3.6: Dies ist eine Martix einer gewichteten gleichverteilten Matrix, bei der übereinstimmende Symbole mit dem Parameter 0.2 gewichtet ist. Auf der Kopfzeile und der Vorspalte sind die fünf Symbole A, C, G, T und GAP für Lücke abgebildet.

Aufgabe

Die Aufgabe besteht darin nachzuweisen, dass die implementierte Evolutionsstrategie das GPHMM so trainiert, dass die Genvorhersage verbessert wird.

Versuchsaufbau

Es wurden vier Testszenarien mit drei unabhängigen, zufällig initialisierten Läufen untersucht. Dabei wurden für jedes der vier Szenarien der beschriebene (1+1)-EA mit drei unterschiedliche Initialisierungen des Zufallsgenerators und jeweils 10.000 Generationen durchgeführt. Die vier Testszenarien unterscheiden sich in der Parameterisierung der Mutationsoperatoren. Es werden drei Parameter benutzt:

- Die Anzahl der Veränderungen, die bei einer Mutation an einem Zustand durchgeführt werden,
- Die Anzahl der Mutationen, die pro Generation durchgeführt werden,

- Die Wahrscheinlichkeit, dass eine Veränderung an einer zufälligen Transition geschieht.

Die Tabelle 3.7 stellt die Ergebnisse dar. Bei den zu Grunde liegenden Testdaten handelt es sich um einen Abschnitt auf dem menschlichen Chromosom 21 von Nukleotid 46143339 bis 46935758, welches mit dem äquivalenten Chromosom der Maus aligniert ist.

Resultate und Verifikation

Die ermittelten Ergebnisse sind in tabellarischer Form in Tabelle 3.7 dargestellt.

Szenario	Seed	MutZ	MutG	TransMut.	Fitness	Vorhersage
1	17	5	5	0%	-680226.83	8%
1	321	5	5	0%	-777791.99	70%
1	666	5	5	0%	-671479.59	0%
2	17	10	5	0%	-675987.38	8%
2	321	10	5	0%	-675777.69	70%
2	666	10	5	0%	-671972.78	0%
3	17	2	5	20%	-678274.34	8%
3	321	2	5	20%	-668692.44	70%
3	666	2	5	20%	-678272.41	0%
4	17	2	5	100%	-675727.26	8%
4	321	2	5	100%	-730918.27	70%
4	666	2	5	100%	-766758.83	0%

Tabelle 3.7: Darstellung der Versuchsergebnisse dar. Für jedes Szenario wurden drei Zufallsgeneratorwerte (Seeds; bezogen auf `java.util.Random`) eingesetzt und getestet. Die Spalten MutZ und MutG geben die Anzahl an Mutationen pro Zustand und die Anzahl an Mutationen pro Generation an. TransMut beschreibt die Wahrscheinlichkeit, dass eine Mutation an den Transitionen durchgeführt wird. In der Spalte Fitness wird der Fitnesswert des besten gefundenen Ergebnisses dieser Variante dargestellt. In der letzten Spalte Vorhersage ist der Wert eingetragen, der der Vorhersage des besten Gens des optimierten Pfades entspricht. Diese Vorhersage wurde mit dem „Nucleotide-Nucleotide BLAST"-Algorithmus ermittelt.

Beobachtungen

An den Ergebnissen der Tabelle zeigt sich, dass die jeweiligen Läufe eines Szenarios zu sehr unterschiedlichen Ergebnissen führen. Nur in einigen Testläufen war eine Vorhersage möglich. Zudem ist ersichtlich, dass alle vier Szenarien trotz unterschiedlicher Parametrisierung zu sehr ähnlichen Ergebnissen führen, was den Schluss nahe legt, dass der Einfluss der Parameter auf die Operatoren zu schwach bemessen wurde. Entgegen den Erwartungen führt die Transitionsmutation zu keinen Änderungen am Ergebnis.

Das Ergebnis der Ausgangssituation konnte durch die ES nach der Initialisierung nicht wiedergefunden werden. Insgesamt muss festgestellt werden, dass die Fitnessfunktion mit guten Genvorhersagen schlecht korreliert, da eine Verbesserung des Fitnesswertes

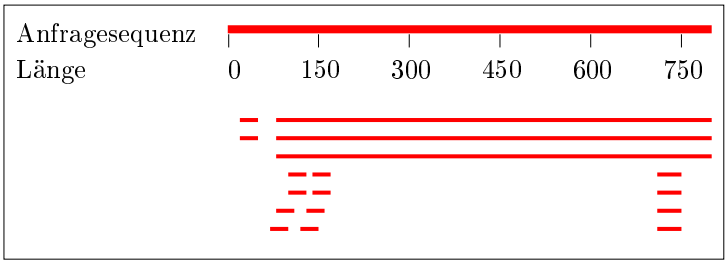

Abbildung 3.24: Die Darstellung zeigt ein Ausgabediagramm des BLAST-Algorithmus, wie es bei einer Verifikation benutzt wird. Für die Genvorhersage der bewerteten Ausgangssituation wurde ein Vergleich mit der Datenbank durchgeführt, der zum dargestellten Ergebnis führt. Der oberste breite Balken stellt die erzeugte Genvorhersage dar. Die darunter befindlichen Balken stellen die Übereinstimmungen mit verschiedenen bekannten Genen dar. Die langen, kurz unterbrochenen, Balken repräsentieren ein gefundenes menschliches Collagen-Gen, welches zu 79% mit der Eingabe übereinstimmt.

erkennbar ist und sich die Vorhersagequalität trotzdem verschlechtert. Die beschriebene Ausgangssituation liefert für eine Bewertung einen Fitnesswert von -751520.21, welcher die logarithmierte Pfadwahrscheinlichkeit darstellt. Unter den vorhergesagten Genen im Pfad befindet sich eine Vorhersage für ein Gen, die zu 79% mit dem Gen für Collagen Typ VI Alpha 1 übereinstimmt (siehe Abbildung 3.24).

Diskussion

Für die getesteten Szenarien wurden, wie bereits beschrieben, keine zufrieden stellenden Ergebnisse erzielt. Da die verwendete (1+1)-Evolutionsstrategie sehr eingeschränkt arbeitet, besteht eine mögliche Verbesserung in der Anwendung anderer (populationsbasierter) Optimierverfahren. Aus den Ergebnissen wird zudem ersichtlich, dass die Operatoren noch nicht optimal an die Problemstellung angepasst sind. Die Transitionsmutation beeinflusst die Ergebnisse kaum. Dies liegt daran, dass die Transitionswahrscheinlichkeit nur schwach in die Berechnung der Pfadwahrscheinlichkeit eingeht. Hierbei könnte eine Gewichtung der Emissionswahrscheinlichkeiten durch die Transitionswahrscheinlichkeit Abhilfe schaffen. Außerdem neigt der Optimierungsalgorithmus dazu, die Genvorhersage auf ein einzelnes Exon hin zu optimieren. Dies liegt daran, dass die Optimierung des Single-Exon-Zustandes des GPHMM einfacher ist, als die Genvorhersage mit Introns und internen Exons zu verbessern. Eine Anpassung der Mutation der Emissionen durch eine Gewichtung könnte die Vorhersage verbessern.

3.5 Strukturvorhersage

Die Sekundärstrukturvorhersage (Protein Structure Prediction, kurz PSP) behandelt das Problem, die unbekannte Sekundärstruktur (siehe Kapitel 2.1) eines Proteins zu bestimmen. Auf Grund beschränkter Ressourcen kann es schwierig sein, eine Bestimmung akkurat durchzuführen. Zum einen ist der materielle Aufwand bei genauen Untersuchungen im Labor sehr hoch, zum anderen sind Rechnerkapazitäten zu gering, als dass eine exakte Vorhersage in akzeptabler Zeit berechnet werden könnte.

Ein Versuch, diese Probleme zu umgehen, aber trotz Allem eine brauchbare Annäherung zu erhalten, ist der Einsatz von Evolutionären Algorithmen. Ein solcher Ansatz wurde in der PG487 entwickelt und soll hier vorgestellt werden.

3.5.1 Beschreibung des implementierten Ansatzes

Der implementierte Ansatz basiert auf der Optimierung spezieller HMM mittels einer ES (siehe Kapitel 2.5.1). Basierend auf einer Vermutung über Aminosäuren in Martin et al. [MGR05] und einem Ansatz zur Lösung von Won et al. [WHPBK05] konnte eine Idee für ein Programm entwickelt werden.

3.5.1.1 Verwendete Literatur

In der Arbeit von Martin et al. [MGR05] wird davon ausgegangen, dass Teile einer Aminosäuresequenz, die eine bestimmte Struktur ausbilden (Helix, Faltblatt und Coil), eine Auffälligkeit (ein Muster) der hydrophob/polar-Eigenschaften (siehe Kapitel 2.1) ihrer Aminosäuren aufweisen. Es können, in Anlehnung an eine bestimmte Struktur, bestimmte Motive oder Muster gefunden werden. Anhand dieser Muster wird durch ein HMM die Sekundärstruktur bestimmt.

So kommt zum Beispiel das Motiv *pphh* oder *hhpp*, wobei p für polar und h für hydrophob stehen, zu 69% in den Teilen der Sequenzen vor, die eine Helix bilden. Anzumerken wäre noch, dass das HMM in der Arbeit von Martin et al. [MGR05] manuell erstellt wurde. Der Unterschied wird deutlich, wenn der Ansatz von Won et al. [WHPBK05] damit verglichen wird.

In der Arbeit von Won et al. [WHPBK05] wird ein GA (siehe Kapitel 2.5) verwendet, um mit dessen Hilfe ein HMM zu optimieren, welches die Sekundärstrukturen bestimmen soll. Die HMM werden randomisiert erzeugt und von dem GA manipuliert, bis ein gewünschtes bzw. optimales HMM erstellt ist.

3.5.1.2 Ansatz der Implementierung

Der gewählte Ansatz zur Vorhersage von Protein-Sekundärstrukturen lässt sich folgendermaßen beschreiben: Es werden zunächst aus Datensätzen bekannter Proteine häufig auftretende Motive der einzelnen Strukturen bestimmt. Aus diesen Motiven werden dann mehrere HMM gebildet, im folgenden Motiv-HMM genannt, welche ausschließlich Sequenzen emittieren können, die dem Motiv entsprechen. Für jede Struktur existieren dann eine Vielzahl von Motiv-HMM.

Anschließend werden diese Motiv-HMM zu mehreren großen HMM zusammengesetzt. Diese Sammlung von HMM wird dann mit dem ES optimiert, bis am Ende ein einziges, annähernd optimales HMM für das PSP gefunden wird. Mit diesem sollten dann beliebige Proteine bestimmt werden.

3.5.1.3 Labeled-HMM

HMM können gewöhnlich nur ein einzelnes Symbol eines Alphabets emittieren. Da jedoch zusätzlich zur emittierten Aminosäuresequenz die Sekundärstruktur bestimmt werden soll, werden „Label“ eingeführt. Jedem Zustand in diesen Labeled-HMM wird dann genau ein Label zugeordnet, welches selbst ein Symbol eines Alphabets ist. In diesem Fall besteht das Alphabet der Label aus den drei Sekundärstrukturen, die vorhersagt werden sollen: Helix, Faltblatt und Coil.

Um nun die Sekundärstruktur einer Aminosäuresequenz mittels eines solchen HMM vorherzusagen, wird mit dem Viterbi-Algorithmus (vergleiche Kapitel 2.4.1) die wahrscheinlichste Zustandsfolge berechnet, welche die Sequenz emittieren würde. Aus dieser Zustandsfolge können nun die Label der Zustände extrahiert werden: Ist der Viterbi-Pfad beim Emittieren der Aminosäure an Position t in Zustand i, so wird das Label von Zustand i, $Label(i)$, als Sekundärstruktur in Position i angesehen. Für diese Änderung der HMM muss der Baum-Welch-Algorithmus (vergleiche Kapitel 2.4.2) jedoch angepasst werden. Die Beschreibung dazu befindet sich in Kapitel 3.5.3.6.

3.5.1.4 Der Q-Score

Der Q-Score ist eine Funktion, mit der es möglich ist, eine Strukturvorhersage zu bewerten. Er wird in den meisten Veröffentlichungen, die sich mit dem Problem der Sekundärstrukturvorhersage beschäftigen, verwendet. Dies erlaubt einen Vergleich mit Ergebnissen anderer Arbeiten mit den hier erzielten Resultaten. Der Q_i-Wert ist der prozentuale Wert der korrekt bestimmten Residuen (d.h. Aminosäuren bei Proteinen) als Helix (Q_H), Faltblatt (Q_S), Coil (Q_C) oder für alle drei Strukturen (Q_3). Er lässt sich wie folgt berechnen:

- Q-Score für einzelne Zustände:

$$Q_i = \frac{\text{Anzahl der Residuen, die im Zustand } i \text{ korrekt bestimmt wurden}}{\text{Anzahl der Residuen, die im Zustand } i \text{ beobachtet werden}} \cdot 100,$$

- Für alle drei Zustände:

$$Q_3 = \frac{\text{Anzahl der Residuen, die korrekt bestimmt wurden}}{\text{Anzahl aller Residuen}} \cdot 100.$$

3.5.2 Formalien

3.5.2.1 DSSP - Die Datensätze

DSSP (Dictionary of Secondary Structure of Proteins) ist eine Datenbank von Proteinen, deren Sekundärstruktur bestimmt wurde. Sie soll dabei helfen, ein einheitliches Format

für bekannte Strukturen zu etablieren. Das Dateiformat besteht aus einer tabellarischen Beschreibung der Aminosäuren im Textformat. In jeder Zeile werden die Informationen der nächsten Aminosäure beschrieben. Neben genaueren Beschreibungen, wie dem Winkel zwischen den Atomen, gibt es auch die Zuordnung der Struktur an, in der sich das Residuum befindet. Diese Zuordnung kann eine der folgenden sein:

H: α-Helix,

B: Isolierte β-Bridge,

E: Extended Strand (Faltblatt),

G: 3-Helix (3/10-Helix),

I: 5-Helix (π-Helix),

T: Hydrogen Bonded Turn,

S: Bend,

_: Coil.

Um die Vorhersage zu vereinfachen, wurden von Martin et al. [MGR05] nur die drei Klassen Helix (H), Faltblatt (E) und Coil (C) vorhergesagt. Dazu werden H und G des Formats auf H reduziert, E und B auf E und alle übrigen auf C.

3.5.2.2 Anforderungen an die Implementierung

Die spezifischen funktionalen Anforderungen, die an das Programm gestellt wurden, waren bezüglich der Eingabe an das System, dass Datensätze im DSSP-Format (ein Protein) benutzt werden sollten, des Weiteren sollte als Ausgabe die Sekundärstrukturen eines Proteins als Sequenz vorliegen. Als externe Softwareschnittstelle wurden die Algorithmen der HMM-Implementierung aus Kapitel 3.1 verwendet.

3.5.2.3 Ziel der Implementierung

Das Ziel dieser Arbeit war die Implementierung eines (sekundär) Proteinstruktur-Vorhersage Systems/Programms, auf Basis eines HMM mit Beteiligung einer ES. Dieses System sollte die Vorhersage aller Strukturen beinhalten und einen Q_3-Score von mindestens 50% erreichen, eine rein zufällige Vorhersage kommt auf einen Wert von 34.5%.

3.5.2.4 GUI

Die GUI (siehe Abbildung 3.25) dieses Systems gibt einem Benutzer die Möglichkeit, ein oder mehrere HMM und beliebige Proteine auszuwählen. Durch die GUI ist es dann möglich, die Vorhersage zu starten und vor allem eine adäquate Ausgabe zu generieren, die die Proteinsequenz, Struktur, sowie die erzielten Q-Scores enthält.

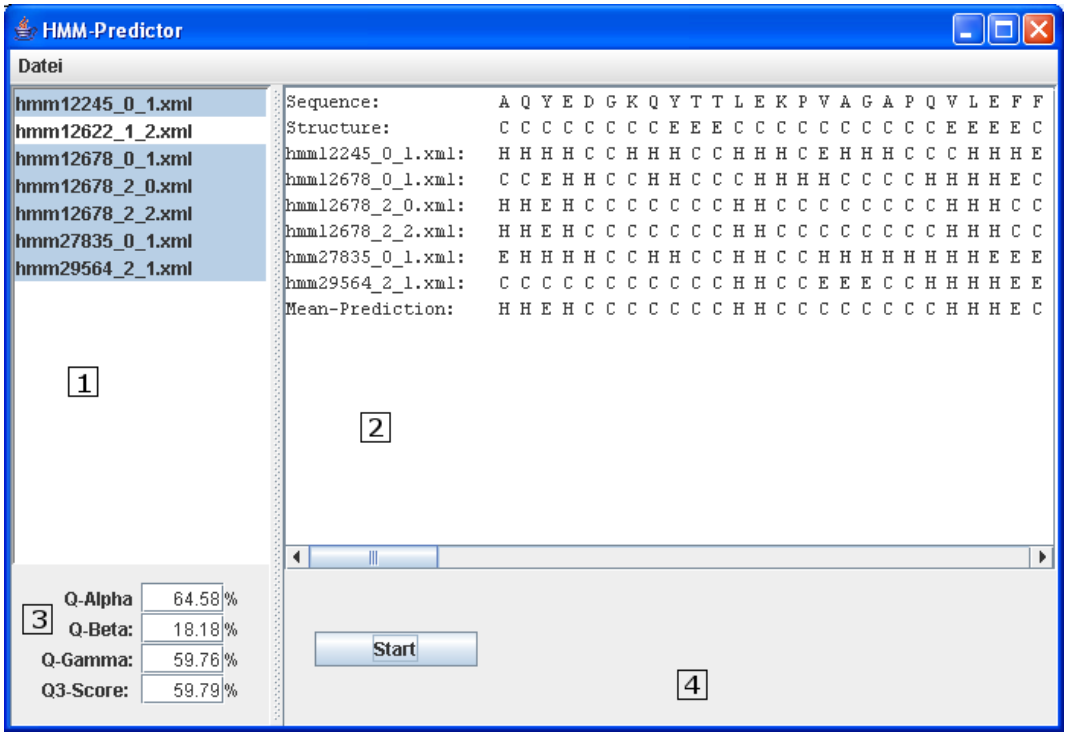

Abbildung 3.25: Nachdem der Benutzer verschiedene HMM und DSSP-Dateien, unter dem Menüpunkt „Datei“, ausgewählt hat, sind diese HMM unter 1 aufgezählt und beliebig anwählbar. Unter Punkt 2, dem Hauptfenster, werden die Proteinsequenz ihre korrekte Struktur, sowie die verschiedenen vorhergesagten Strukturen der gewählten HMM und die dadurch entstehende Gesamtvorhersage aller HMM, angezeigt. Bei Punkt 3 sind die berechneten Q-Scores einer angewählten Vorhersage aus dem Hauptfenster dargestellt. Der Button mit der Aufschrift „Start“, unter 4, startet die Vorhersage mit den gewählten HMM und DSSP-Dateien.

3.5.3 Die ES und ihre Operationen

3.5.3.1 ES-Paket Dortmund

Um dem Programm einen erprobten ES-Rahmen zugrunde zu legen wurde entschieden, das ES-Paket der Universität Dortmund (siehe Kapitel 2.6.1) zu verwenden. Es mussten die Klassen für die Fitnessfunktion, Individuen, Population, Rekombination und Mutation angepasst werden, da das Problem in dem Paket nicht vorgesehen ist.

3.5.3.2 Der SecStrucDealer

Die Klasse `SecStrucDealer` stellt Sequenzen von Proteinen bereit, die in einem Ordner als DSSP-Dateien hinterlegt wurden. Der Konstruktor der Klasse prüft, ob die Zahl der angegebenen Dateien mit denen im Ordner übereinstimmen und ob diese auch DSSP-Dateien sind.

3.5.3.3 Fitnessfunktion

Als Fitnessfunktion wird der negative Q-Score verwendet, da das Paket auf Minimierungsprobleme zugeschnitten ist. Dazu wird vom `SecStrucDealer`-Objekt eine Sequenz angefordert, auf die das Individuum hin getestet wird. Auf diese Sequenz wird das HMM angewandt und die resultierende Sekundärstruktur mit der korrekten Struktur der Sequenz verglichen. Um eine genauere Aussage über die Güte des HMM zu machen, werden 4 Auswertungen durchgeführt und der Mittelwert als Fitness des Individuums übernommen. Die für die Fitness zuständige Klasse ist `secondarystructure.ea.problem.Q`.

3.5.3.4 Individuen und Population

IndividualComposedHMM und PopulationComposedHMM

Im Paket `secondarystructure.ea.population` sind die für die Population nötigen Klassen implementiert. Die Klasse `IndividualComposedHMM` implementiert das Interface `population.IIndividual` des ES-Rahmenpakets. `PopulationComposedHMM` erbt von der Klasse `population.Population` und erweitert diese um die Methode `initializePopulation(SecStrucDealer dealer)` zur Initialisierung der Individuen, welche beim Start der ES aufgerufen wird. Dies erfolgt wie in Kapitel 3.5.1.2 beschrieben. So werden zunächst Sequenzen vom `SecStrucDealer`-Objekt angefordert und auf häufig auftretende Motive hin untersucht. Dies geschieht in der Klasse `secondarystructure.motiffinder.PatternUtil`. Die dabei gefundenen Motive haben eine Länge $2 \leq l_{Motiv_i} \leq 6$. Von diesen Motiven werden dann für jede der drei Sekundärstrukturen Helix, Faltblatt und Coil die Motive mit der besten Güte gewählt. Die verwendete Güte wird nach folgender Formel berechnet:

$$Güte_{Motiv_i} = a_{Motiv_i} \cdot l_{Motiv_i}^{0,25},$$

wobei a_{Motiv_i} der Häufigkeit des Auftretens des Motivs i in der durchsuchten Sequenz entspricht. Diese Formel wurde gewählt, um größere Muster kleineren vorzuziehen. Jedoch sollten keine zu großen Motive bevorzugt werden, was durch den Exponenten abgeschwächt wird. Die Anzahl der verwendeten Motive pro Sekundärstruktur wird durch

$$\min\{n_{Motive,struc}, 4 \pm 2\}$$

bestimmt, wobei $n_{Motive,struc}$ die Anzahl der gefundenen Motive für die Sekundärstruktur $struc \in \{Helix, Faltblatt, Coil\}$ bezeichnet. Aus diesen Motiven werden dann von der Klasse `secondarystructure.hmm.ModuleHMMBuilder` Motiv-HMM erstellt, welche nur Transitionen mit Gewicht 1 beinhalten.

Abschließend muss aus den einzelnen erzeugten Motiv-HMM ein gesamtes HMM zur Strukturvorhersage erstellt werden. Dies erfolgt, indem vom letzten Zustand jedes Motiv-HMM eine Transition zum ersten Zustand eines jeden Motiv-HMM führt. Die Gewichte dieser Transitionen sind dabei gleichverteilt randomisiert. Das nun erhaltene HMM ist vom Typ `secondarystructure.hmm.ComposedHMM`, welches von `hmm.SimpleHMM` erbt und zusätzlich Zeiger auf die jeweiligen Anfangs- und Endzustände der einzelnen Motiv-HMM hält. Es wird dann einem Individuum zugewiesen und mit der Erstellung des nächsten HMM fortgefahren bis die Population vollständig initialisiert ist.

3.5.3.5 Rekombination

Die Rekombination wird mit einer Wahrscheinlichkeit p_{recom} aufgerufen. Es wird dazu ein Anteil von p_{recom} der Individuen der Nachkommen durch Rekombination erstellt. Die restlichen Nachkommen werden durch Klonen eines vorhandenen Individuums erzeugt.

Bei der Rekombination werden zwei Individuen miteinander kombiniert, um ein Neues zu erstellen. Dazu werden die drei Label-Klassen der Zustände (Helix, Faltblatt, Coil) gemischt. Es wird für das erste Elternindividuum gleichverteilt randomisiert entschieden, welches seiner drei Teile in das neue Individuum eingehen soll. Vom zweiten Elternindividuum gehen dann die Zustände mit den Labels der anderen zwei Klassen ein. Alle Transitionen unter den Zuständen, die von einem Elternindividuum übernommen werden, bleiben dabei erhalten. Es fehlen dann noch die Transitionen zwischen den Zuständen des ersten und denen des zweiten Elternindividuums. Diese werden gleichverteilt randomisiert gewichtet. Da die Transitionen der Elternindividuen übernommen werden, wird aber der Bereich, in dem sich die Wahrscheinlichkeiten der restlichen Transitionen befinden, festgelegt:

$$\sum_{(i,j)\in M_1\times M_2} a_{i,j} = 1 \quad - \sum_{(i,j)\in M_1\times M_1} a_{i,j}$$

wobei M_i die Menge der Zustände vom Elternindividuum i definiert.

3.5.3.6 Mutation

MutationTrans

Bei der Mutation `MutationTrans` wird für jedes Individuum genau ein Zustand seines HMM mutiert. Bei diesem werden dann die Transitionswahrscheinlichkeiten seiner ausgehenden Transitionen geändert. Der zu mutierende Zustand wird dabei gleichverteilt randomisiert unter den Endzuständen der Motiv-HMM gewählt, denn nur diese haben mehr als eine ausgehende Transition. Für die ausgehenden Transitionen des gewählten Zustandes wird nun eine neue Verteilung generiert, indem zu jeder Transitionswahrscheinlichkeit $t_{i,j}$ der Betrag einer normalverteilten Zufallsvariable $G \sim \mathcal{N}(0,1)$, gewichtet mit einem Faktor η, der die Schrittweite bestimmt, addiert wird:

$$t_{i,j} = t_{i,j} + |G| \cdot \eta.$$

Anschließend werden die Transitionswahrscheinlichkeiten wieder normiert:

$$t_{i,j} = \frac{t_{i,j}}{\sum_k t_{i,k}}.$$

MutationBW

Die Mutation `MutationBW` wird nur aufgerufen, falls das Individuum während dieses Schleifendurchlaufs durch Rekombination erstellt wurde. In diesem Fall werden zwei Iterationen einer modifizierten Baum-Welch-Optimierung durchgeführt. Bei diesen Iterationen werden wieder Sequenzen vom `SecStrucDealer`-Objekt angefordert, mit denen der Baum-Welch-Algorithmus (vergleiche Kapitel 2.4.2) gestartet wird.

Die Modifikationen des Baum-Welch-Algorithmus beschränken sich auf zwei Änderungen. Zum einen muss neben der Optimierung des HMM zur Emission der Sequenz auch darauf geachtet werden, dass das HMM sich beim Emittieren des Buchstaben an Position j in einem Zustand befindet, welcher ein Label trägt, dass der Sekundärstruktur an der Position j entspricht. Um dies zu unterstützen, wurde eine Bestrafung implementiert, bei der für Zustand i die Variablen des Forward- und Backward-Algorithmus $\alpha_i(t)$ und $\beta_i(t)$ (siehe Kapitel 2.4.2) folgendermaßen geändert werden:

$$\alpha_i(j) = \begin{cases} \alpha_i(j) & \text{falls } \mathit{Label}(i) = \text{Sekundärstruktur(j)} \\ \alpha_i(j) \cdot 0,6 & \text{sonst} \end{cases}.$$

Die zweite Modifikation betrifft die Änderung der Emissions- und Transitionswahrscheinlichkeiten. Um ein Übertraining des HMM auf die gewählte Sequenz zu vermeiden, werden die vom Baum-Welch-Algorithmus vorgeschlagenen neuen Wahrscheinlichkeiten nicht unverändert übernommen, sondern die neue Transition setzt sich zu 30% aus der alten Wahrscheinlichkeit, sowie zu 70% aus der neuen zusammen.

3.5.4 Ergebnisse

Experiment: Training von HMM mittels EA zur Vorhersage der Sekundärstruktur von Proteinen

Vorexperimentelle Planung

Zur Optimierung der Parameter der ES wurde das SPOT-Programm (siehe Kapitel 2.6.3) verwendet. Zunächst wurde die Anzahl der in der SPO zu optimierenden Parameter eingegrenzt. Dazu wurden zunächst die Parameter der Baum-Welch-Mutation festgesetzt, indem mittels einmaliger Baum-Welch-Mutation auf einzelnen Individuen die Änderung des Q-Score gemessen wurde. Dabei stellte sich eine Anzahl an Aufrufen des Baum-Welch-Algorithmus von 2 als beste Einstellung heraus. Außerdem konnte für eine Bestrafung wie in Kapitel 3.5.3.6 der Wert $0,6$ als gut bestimmt werden. Durch die starke Varianz der Ergebnisse kann hier allerdings nicht von einem Optimum gesprochen werden. Weiterhin wurde die Stärke der Transitionsmutation festgelegt. So wurde die Anzahl der zu mutierenden Zustände pro Mutation auf 1 und der Faktor zur Stärke der Änderung $\eta = 0,1$ gesetzt. Diese Festlegungen wurden nicht experimentell gewonnen, sondern durch subjektive Einschätzung gewählt.

Aufgabe

Die Aufgabe besteht darin, ein HMM zu entwickeln, welches in der Lage ist Sekundärstrukturen von Proteinen so zu bestimmen, dass ein Q_3-Wert von mindestens 50% erreicht wird.

Versuchsaufbau

Neben den Parametern sind auch die Größe der Elternpopulation $\mathbb{P}_p$, die Anzahl der

Nachkommen $\mathbb{P}_o$, sowie die Rekombinationswahrscheinlichkeit p_{recom} zu wählen. Zur Optimierung dieser Parameter wurde SPOT mit 4 Schritten zu je 30 Designpunkten gestartet, wobei jeder Designpunkt in 3 Iterationen der ES getestet wurde. Aufgrund der langen Laufzeit von ca. 3 Stunden pro Designpunkt konnten nicht mehr als 4 Schritte in SPOT durchgeführt werden. Bei Tests mit einer anderen ES konnten jedoch mit 4 Schritten schon gute Modelle zur Wahl der Parameter erstellt werden. Als Testdatensätze wurden 100 der DSSP-Dateien von Kabsch und Sander [KS05] gewählt.

	SPO-Schritt				
	1	2	3	4	Gesamt
Maximum	67,47	67,03	67,89	68,72	68,72
Minimum	41,30	50,01	40,90	51,10	40,90
Durchschnitt	57,98	58,33	58,23	59,02	58,39
Standardabweichung	4,08	2,96	4,10	3,52	3,70

Tabelle 3.8: Angabe der Maxima, Minima, Durchschnittswerte und Standardabweichung des erzielten Q_3-Score während der einzelnen SPO-Schritte, sowie über die gesamte SPO.

Resultate

In Tabelle 3.8 sind zu den einzelnen Schritten der SPO, sowie über die gesamte Optimierung, Maximum, Minimum, Durchschnitt und Standardabweichung einzusehen. Eine Veranschaulichung der Ergebnisse der gesamten SPO als Graph zeigen Abbildungen 3.26, 3.27 und 3.28.

Beobachtung

Aus Abbildung 3.26 lassen sich Empfehlungen für die Parametereinstellungen gewinnen. So weist der Graph auf einen Wert von $\mathbb{P}_p \approx 14$ hin. Weiterhin scheint $p_{recom} \approx 0,4$ eine gute Wahl zu sein. Dieser Wert erscheint jedoch ungewöhnlich hoch und lässt sich dadurch erklären, dass die durch Rekombination erstellten Individuen durch die Baum-Welch-Mutation (siehe Kapitel 3.5.3.6) lokal optimiert werden. Es lässt sich dann aber die Wirkung der ES zur Suche nach einem globalen Optimum anzweifeln. Der Graph zum Verhältnis von $\mathbb{P}_o/\mathbb{P}_p$ kann leider zu keiner Aussage führen. Zum einen ist der Bereich der Funktionswerte mit 57,5 – 59 sehr eingeschränkt, zum anderen weist der Kurvenverlauf nur undeutlich auf Optima hin. Weiterhin zeigt sich in Abbildung 3.27 eine Varianz der Ergebnisse, die keinen erkennbaren Regeln folgt.

In Abbildung 3.26 lässt sich außerdem erkennen, dass die tatsächlichen Ergebnisse teilweise stark um das Modell streuen. Diese starke Varianz lässt sich darauf zurückführen, dass aufgrund der langen Laufzeit nur vier Auswertungen pro Individuum durchgeführt werden. Dazu werden, wie in Kapitel 3.5.3.3 beschrieben, zufällig Sequenzen aus einem Pool gewählt um eine Optimierung auf alle Sequenzen des Pools zu fördern. Dies zieht allerdings das Problem nach sich, dass einzelne Individuen auf ähnliche Sequenzen ausgewertet werden und dabei einen guten Q_3-Score erreichen, jedoch nicht universell einsetzbar sind. Da später keine weitere Auswertung dieser Individuen erfolgt bleiben diese als gute Lösung erhalten. Dieser Einfluss des Zufalls könnte die starke Streuung und den

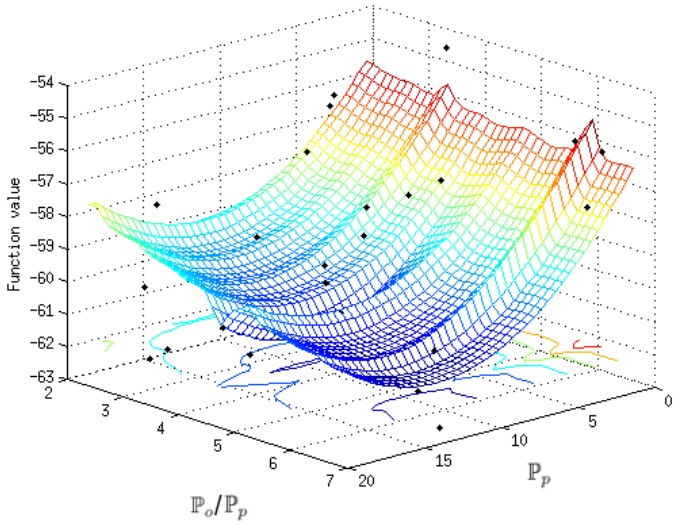

Abbildung 3.26: Die Parameter $\mathbb{P}_p$ und das Verhältnis $\mathbb{P}_o/\mathbb{P}_p$ sind gegen die Funktionswerte (Function value) aufgetragen. Der Graph zeigt das durch SPOT erstellte Modell des Suchraums, wobei die Farben den Funktionswerten zugeordnet sind. Schwarzen Punkte kennzeichnen die tatsächlichen Ergebnisse. Es lässt sich leicht erkennen, dass diese stark streuen.

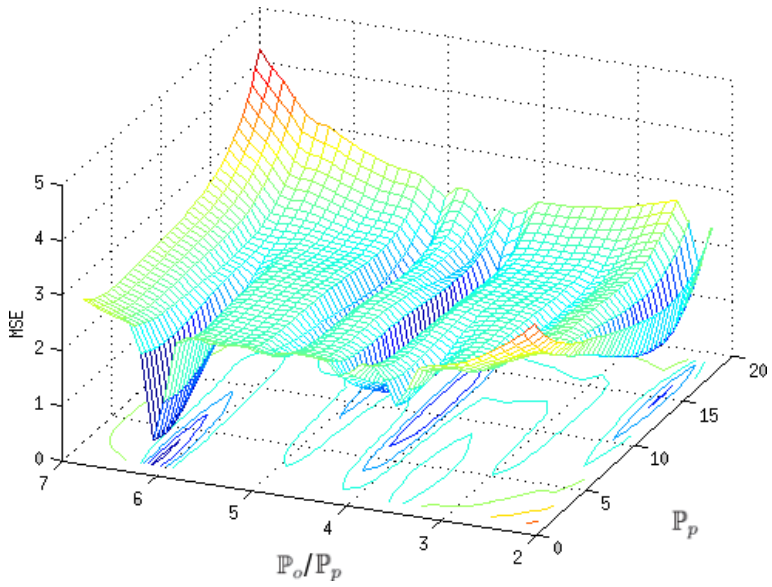

Abbildung 3.27: Die Parameter $\mathbb{P}_p$ und das Verhältnis $\mathbb{P}_o/\mathbb{P}_p$ sind gegen die Standardabweichung (MSE) aufgetragen. Gezeigt wird hier das von SPOT erstellte Modell des stochastischen Prozesses. Auch hier zeigt sich die Unsicherheit des Modells, da sich keine den Parametern angepasste Struktur erkennen lässt.

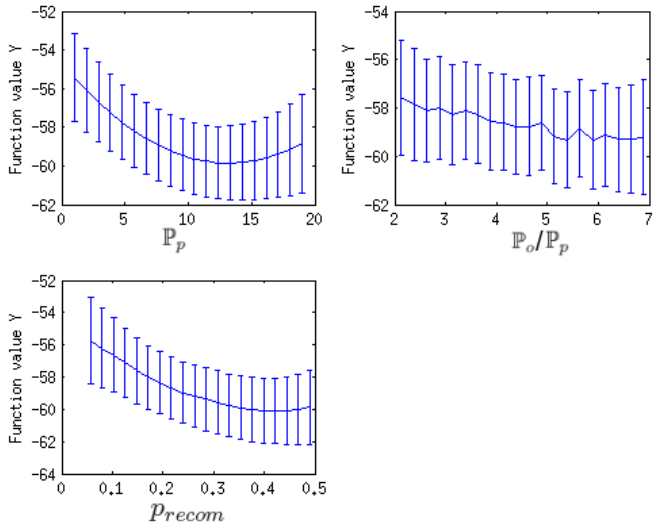

Abbildung 3.28: Die zu optimierenden Parameter sind gegen den Funktionswert aufgetragen. Es lässt sich aus dem Graph von $\mathbb{P}_p$ ein Wert von 14 schließen. Die Rekombinationswahrscheinlichkeit p_{recom} weißt auf einen Wert von $0,4$ hin. Das Verhältnis $\mathbb{P}_o/\mathbb{P}_p$ kann aufgrund des undeutlichen Verlaufs und der geringen Änderung des Funktionswertes nicht ausreichend bestimmt werden.

undeutlichen Verlauf des Modells für $\mathbb{P}_o/\mathbb{P}_p$ erklären. Weiterhin könnten mehr Schritte der SPO zu einer Verbesserung des Modells führen.

Diskussion

Zusammenfassend kann gesagt werden, dass dieser Ansatz in der Lage ist, positive Ergebnisse zu erbringen. Ob allerdings dieser Ansatz mit heutigen State-of-the-art-Programmen konkurrieren kann, steht aus. Bei diesem Projekt war der Faktor Zeit ein Problem, das am Ende das Programm in seiner jetzigen Form nicht konkurrenzfähig macht mit aktuellsten Programmentwicklungen anderer Forscher/Institute. Das Programm besitzt sicherlich Potential, höhere Q-Werte zu erreichen. Ansätze um ein Maximum aus diesem Programm zu holen, ist eine tiefgehendere Auseinandersetzung mit molekularbiologischen Themen zur Proteinfaltung. Interessant wäre zudem eine Betrachtung aller Strukturklassen, nicht nur die Reduktion auf Helix, Faltblatt und Coil, und die Aufnahme weiterer chemischer Eigenschaften von Aminosäuren in den Algorithmus, um die Vorhersage stabiler gegenüber Fehlern zu machen. Bezüglich der Erfüllung der Projektziele kann gesagt werden, dass diese erreicht wurden. Das Programm ist lauffähig und der aktuell ermittelte Q_3-Score liegt bei über 50%, dem gesetzten Ziel.

3.6 Phylogenetische Bäume und Clustering

3.6.1 Phylogenetische Bäume

Unter Phylogenie (dt. Stammesgeschichte) ist die Beschreibung biologischer Verwandtschaftsbeziehungen zwischen verschiedenen Arten, Proteinen und Ähnlichem zu verstehen. Oft wird als Darstellung dieser Beziehungen ein Graph mit Knoten und Kanten zwischen diesen benutzt. Dieser Graph ist zusammenhängend und kreisfrei, d.h. zwischen zwei Knoten gibt es stets genau einen Weg. Enthält der Graph nur gerichtete Kanten und wird ein Knoten ausgewählt, von dem aus alle anderen Knoten erreicht werden können, handelt es sich um einen Baum und dieser Knoten ist die Wurzel. Verlassen ferner jeden inneren Knoten genau zwei gerichtete Kanten, handelt es sich um einen binären Baum. In einem phylogenetischen Baum genügen aus einem Knoten zwei ausgehende Kanten, wenn davon ausgegangen wird, dass neue Arten, Proteine o.ä. in der Weise entstehen, dass sich eine Spezies in genau zwei Richtungen entwickelt. An den Blättern des Baumes befinden sich die heute beobachtbaren Arten, jeder innere Knoten steht für angenommene Vorgänger. Da die gerichteten Kanten eine (nicht quantifizierte) zeitliche Entwicklungsrichtung wiederspiegeln, symbolisiert die Wurzel den gemeinsamen Vorfahren aller im Baum enthaltenen Individuen (siehe auch [Les03]).

3.6.2 Clustering

Mit Clustering ist allgemein die Partitionierung einer Menge von Objekten anhand ihrer Ähnlichkeit in disjunkte Klassen gemeint. Die Objekte (Arten, Proteine, DNA-Sequenzen etc.) ähneln anderen Objekten ihrer jeweiligen Klasse in Hinblick auf ein zu wählendes Ähnlichkeitsmaß stärker als Objekten außerhalb dieser Klasse. Es gibt eine Reihe von Methoden, mit Hilfe derer sich Ähnlichkeit zwischen Objekten quantifizieren lässt. Innerhalb dieser Methoden bieten sich oft verschiedene Möglichkeiten bzw. Kriterien, nach denen sich ein Wert für die Ähnlichkeit ermitteln lässt. Je robuster das Ergebnis eines Clustering gegenüber der Wahl verschiedener Kriterien ist, desto verlässlicher wird es eingeschätzt.

Sind als Elemente eines Clusters nicht nur die ursprünglichen Objekte, sondern Subcluster zugelassen, handelt es sich um ein hierarchisches Clustering. Als ein wichtiges Verfahren soll die so genannte „UPGMA-Methode“ erläutert werden.

3.6.3 Die UPGMA-Methode

Die Bildung eines phylogenetischen Baumes mittels UPGMA (Unweighted Pair Group Method with Arithmetic mean, siehe [SM58]) erfolgt mit Hilfe von Multiplen Sequenzalignments. Dabei wird von einer konstanten Evolutionsrate ausgegangen. Das heißt, dass sich alle Arten in gleichem Tempo verändern. Ein Multiples Sequenzalignment wird benötigt, da es zur Abstandsfindung auf Sequenzen bei Wörtern unterschiedlicher Länge keinen Anhaltspunkt gibt, der ein Messen ermöglicht. Multiple Sequenzalignments stellen zum einen sicher, dass alle Sequenzen gleiche Länge haben, zum anderen sind Wortstücke großer Ähnlichkeit einander zugeordnet, so dass hier ein Anhaltspunkt zur

Abstandsfindung entsteht. Zur Distanzfindung werden die paarweisen Abstände aller Sequenzen zueinander in eine Distanzmatrix geschrieben. Zur Bestimmung der Distanz zweier Sequenzen wird eine dem Hamming-Abstand analoge Methode verwendet. Dieser Abstand wird definiert durch die Anzahl an Stellen, an denen sich zwei Sequenzen unterscheiden. Es können ebenso unterschiedliche Gewichtungen für die Abstände zweier Zeichen definiert werden, so dass ein Abstand C zu G mit 1 bewertet wird, während G zum Gap-Symbol als größere Abweichung modelliert wird. In Matrixelement (i, j) wird der jeweilige Abstand d_{ij} eingetragen. Nun erfolgt die eigentliche UPGMA-Berechnung, um die Distanzmatrix zu clustern.

Dazu werden sämtliche Elemente, die geclustert werden sollen, einem Cluster zugewiesen. Die beiden Cluster mit der geringsten Distanz zueinander werden zu einem neuen Cluster zusammengefasst. Die Distanzen aller alten Cluster zum neuen Cluster werden ermittelt. Die Abstände der alten Cluster zum neuen Cluster ergeben sich aus den Durchschnittswerten der alten Cluster zu den einzelnen alten Clustern. Für einen neuen Cluster C_k, welcher aus C_g und C_h entstand, und einen vorher und hinterher vorhandenen Cluster C_i gilt also:

$$d_{ik} = \frac{1}{2} \cdot (d_{ig} + d_{ih}). \tag{3.8}$$

Die Schritte zur endgültigen Clusterdarstellung sind in Algorithmus 7 dargestellt. Dabei

```
for alle Cluster do
    bestimme Distanz aller Cluster zueinander;
    Suche zwei Cluster i,j mit geringstem Abstand;
    Speichere diese zwei Cluster i,j;
    if mehr als 2 Cluster vorhanden then
        Beginne erneut;
    end
end
Vereinige die letzten beiden Cluster zu einem Cluster;
```

Algorithmus 7 : Für alle Cluster werden die Distanzen zueinander ermittelt. Dabei werden die zwei Objekte mit geringstem Abstand zu einem Cluster vereinigt. Diese Vorgang wird solange iteriert, bis nur noch zwei Cluster übrig sind. Nach Zusammenfassen dieser beiden Cluster ist der Vorgang beendet.

entsteht ein hierarchischer Baum, welcher sich schriftlich im Newick-Format widerspiegelt. Die Blätter dieses Baumes stellen die einzelnen Spezies dar, und dessen innere Knoten repräsentieren jeweils einen gemeinsamen Vorfahren der Kinder dieses Knotens. Das Newick-Format beschreibt einen Knoten des Baumes mit einer Klammer, während die beiden Kinder des Knotens von Kommata getrennt in den Klammern stehen. Dabei sind Schachtelungen möglich, um die Hierarchie zu beschreiben. Eine beispielhafte Darstellung des Formats sei die folgende, wobei Abbildung 3.29 den dazugehörigen phylogenetischen Baum realisiert:

$$(((a,b),c),(d,e))$$

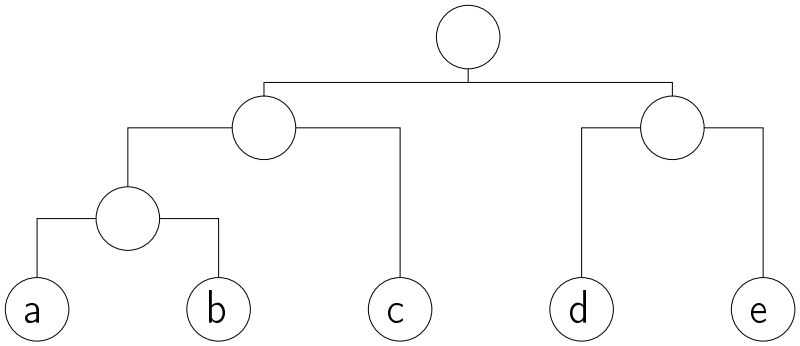

Abbildung 3.29: Der phylogenetische Baum zu der Newick-Darstellung (((a,b),c),(d,e)).

Üblicherweise werden runde Klammern benutzt, obgleich es sich hier um Mengen handelt. Daher gibt es für ein hierarchisches Clustering auch verschiedene Newick-Darstellungen.

3.6.4 UPGMA zur Bestimmung von phylogenetischen Bäumen

Wenn auf jeder Ebene des hierachischen Clustering allein die Zusammenfassung zweier Objekte (Klassen oder ursprüngliche Objekte) zugelassen wird und solange iteriert wird, bis sich alle Objekte in einer einzigen Klasse befinden, kann das Ergebnis zur Erstellung eines phylogenetischen Baumes genutzt werden. Jede neu entstehende Klasse entspricht einem inneren Knoten des Baumes, insbesondere kann die letzte geschaffene Klasse mit der Wurzel identifiziert werden. Als Datenstruktur wird das Newick-Format verwendet. Von einer zu clusternden Menge $M = \{X_1, X_2, ..., X_n\}$ ausgehend werden sukzessive je zwei Objekte zusammengefasst, wobei dies Elemente der Menge oder schon entstandene Klassen sein können. Wie oben beschrieben, werden die geclusterten Objekte in runden Klammern geschrieben, also (o_i, o_j) mit $o_i, o_j \in \{X_k, (o_l, X_p), (o_q, o_r)\}$. Die ursprünglichen Elemente X_s mit $1 \leq s \leq n$ bilden in dem phylogenetischen Baum die Blätter, die Klassen o_t mit $n + 1 \leq t \leq 2n - 1$ bilden die inneren Knoten inklusive der Wurzel. Es sind genau $n - 1$ innere Knoten, weil es so viele Vereinigungen gibt, bis sich n Ursprungselemente in einer Klasse befinden (siehe [HD06]).

3.6.5 Problemdefinition und -analyse

Es sollen verwandte Sequenzen (siehe Abbildung 3.30), die in einer oder mehreren Sequenzdateien vorliegen, benutzt werden, um zu einem Clustering und einem phylogenetischen Baum zu gelangen, die die Evolution der Sequenzen qualitativ wiederspiegeln. Wie bereits beschrieben ist dazu ein Ähnlichkeitsmaß erforderlich, das angibt, welche Objekte sich nahe stehen. Auch muss klar sein, wie die Ähnlichkeit zwischen Klassen und gegebenenfalls Klassen von Klassen ermittelt wird.

Es wurde das im letzten Abschnitt beschriebene Verfahren angewendet. Aus den gegebenen paarweisen Abständen zwischen allen Ausgangssequenzen wird eine Matrix (siehe Abbildung 3.31) aufgebaut. In dieser wird der kleinste Abstand (also die größte Ähnlich-

keit) zwischen zwei verschiedenen Sequenzen gesucht, und genau diese Sequenzen werden in einem Cluster zusammengefasst. Nun müssen die beiden Zeilen und Spalten, die zu den zusammengefassten Objekten gehören, durch eine neue Spalte und eine neue Zeile ersetzt werden. Diese repräsentieren die Klasse, in der sich die Objekte nun befinden. Dazu wird eine Methode benötigt, mit Hilfe derer die neuen Abstände von den unveränderten Objekten zu der neuen Klasse bestimmt werden. Sind auch die letzten beiden Zeilen und Spalten der Matrix zusammengefasst, befinden sich alle Ausgangselemente (vielfach geclustert) in einer einzigen Klasse. Für die Newick-Darstellung bedeutet dies, dass sich alle Klammerausdrücke in einer einzigen Klammer befinden. Sie steht für den letzten geschaffenen Knoten des in 3.32 gezeigten phylogenetischen Baums, die Wurzel.

a = A T C G A T C G A T C G
b = A T C C A T C C A T C C
c = A A G C A A G C A A G C
d = G A T T G A T T G A T T

Abbildung 3.30: Die Eingabesequenzen, für die die Hamming-Abstände zur Überführung in eine Distanzmatrix ermittelt werden.

	a	b	c	d
a	-	**3**	9	12
b	**3**	-	6	12
c	9	6	-	9
d	12	12	9	-

	a,b	c	d
a,b	-	**7.5**	12
c	**7.5**	-	9
d	12	9	-

	a,b,c	d
a,b,c	-	**10.5**
d	**10.5**	-

Abbildung 3.31: Die erste Matrix beschreibt die Distanzen der Eingabesequenzen. Die zweite Matrix folgt nach Clustering von a und b, welche den geringsten Hamming-Abstand zueinander haben. Der dritte Schritt folgt analog. Der bereits bestehende Cluster (a,b) wird zu ((a,b),c) vereinigt. Die daraus entstandene Darstellung im Newick-Format lautet dann: (((a,b),c),d)).

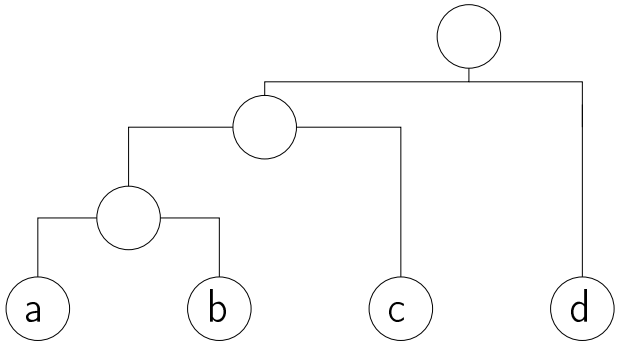

Abbildung 3.32: Der phylogenetische Baum zur Newick-Darstellung (((a,b),c),d)).

3.6.6 Modellierung

Das Hauptproblem ist die Bestimmung des Abstandes zwischen zwei Objekten der Matrix. Zum einen muss dies bei der Berechnung der Ausgangsmatrix geschehen, d.h. es muss der Abstand zwischen den Ausgangssequenzen ermittelt werden. Im Verlauf der Rechnung, wenn neue Klassen von Sequenzen und Klassen von Klassen entstehen, wird eine Methode benötigt, nach der der Abstand zwischen Klassen festgestellt wird. Dabei werden HMM zur Lösung der Aufgabe benutzt. Es folgt eine Beschreibung des Modells, welches die Anforderungen erfüllt.

Zu Beginn werden Sequenzen eingelesen, welche im FASTA-Format vorliegen (siehe Kapitel 2.6.2) und für die ein Clustering und ein phylogenetischer Baum erzeugt werden soll. Für jede Sequenz wird nun ein eigenes HMM erzeugt. Dabei handelt es sich um ein vollständig zusammenhängendes HMM, welchem eine Gleichverteilung zugrunde liegt. Das heißt von jedem Zustand lässt sich jeder Zustand des HMM erreichen, wobei die Transitionswahrscheinlichkeiten alle gleich sind. Die Startwahrscheinlichkeiten, vom Magical-State in einen Zustand des HMM zu gelangen, werden durch $\frac{1}{|S|}$ bestimmt. Die Emissionswahrscheinlichkeiten werden gleichverteilt mit 0,25 für jedes Symbol A, C, G, T in jedem Zustand initialisiert. Die Transitionswahrscheinlichkeiten werden abhängig von der gewünschten Anzahl an Zuständen, abzüglich der Ausgangswahrscheinlichkeit, definiert:

$$t_{ik} = \frac{1 - Ausgangswahrscheinlichkeit}{|S|}.$$

Die Subtraktion der Ausgangswahrscheinlichkeit von 1 ist nötig, da sich die Transitionswahrscheinlichkeiten eines Zustandes sonst zu einer Gesamtwahrscheinlichkeit größer 1 aufsummierten. Die HMM werden auf „ihre“ Sequenz mittels des Baum-Welch-Algorithmus (siehe Kapitel 2.4.2) trainiert, so dass sie diese Sequenz mit großer Wahrscheinlichkeit emittieren. Sie darf allerdings nicht gleich 1 sein, da neben der Trainingssequenz auch die Ausgabe ähnlicher Sequenzen gewünscht ist, und zwar umso wahrscheinlicher, je ähnlicher sie dieser sind.

Im nächsten Schritt werden die Distanzen zwischen allen Paaren ermittelt, um damit die Distanzmatrix zu füllen. Dies geschieht so, dass mittels Forward-Algorithmus der `HMMToolbox` (siehe Kapitel 3.1.4) für alle Sequenzen getestet wird, mit welcher Wahrscheinlichkeit sie von den HMM emittiert werden, die nicht mit ihnen trainiert worden sind. Hohe Emissionswahrscheinlichkeit bedeutet geringe Distanz zu der Sequenz, auf die das HMM trainiert worden ist, eine geringe Wahrscheinlichkeit steht für großen Abstand. Wenn das HMM_i auf die Sequenz X_i trainiert ist, so gibt das Matrixelement $d_{(i,j)}$ die Wahrscheinlichkeit an, mit der das HMM_i die Sequenz X_j emittiert.

Um zu ermitteln, welche zwei Objekte (Sequenzen oder Klassen) in einer Klasse zusammenzufassen sind, müssen die zwei mit geringstem Abstand bestimmt werden. Dabei tritt das Problem auf, dass $d_{(i,j)}$ von $d_{(j,i)}$ verschieden sein kann und in der Regel auch ist. Das HMM_i und das HMM_j emittieren nicht mit derselben Wahrscheinlichkeit die Sequenz, auf welcher das jeweils andere HMM trainiert worden ist. Es liegt keine symmetrische Matrix vor, um zu entscheiden, welches Objektpaar sich am nächsten steht, also muss sowohl $d_{(i,j)}$ als auch $d_{(j,i)}$ betrachtet werden. Es wird das arithmetische Mittel

berechnet, und das Objektpaar mit der größten Ähnlichkeit, also der geringsten Distanz, wird zu einer Klasse zusammengefasst. Ein HMM_k hat die Sequenzen X_i und X_j mit dem in $d_{(k,i)}$ bzw. $d_{(k,j)}$ eingetragenen Wahrscheinlichkeitswert emittiert. Da sich X_i und X_j nun in einer Klasse befinden, ist es sinnvoll zu fragen, wie groß die Wahrscheinlichkeit ist, dass das HMM_k X_i oder X_j emittiert. Das wäre genau $d_{(k,i)}+d_{(k,j)}$ (da die Ereignisse disjunkt sind) und würde zu dem Problem führen, dass der neu geschaffene Distanzwert mindestens doppelt so hoch ist wie einer der Ausgangswerte. Es ist aber nicht einsichtig, warum eine neu geschaffene Klasse näher an alle anderen heranrücken soll, daher wird $d_{(k,i)} + d_{(k,j)}$ durch die Anzahl der beteiligten Klassen geteilt, nämlich 2. Dieser Wert wird an die neue Stelle $d_{(k,l)}$ geschrieben, wenn X_l die neue Klasse bezeichnet.

Die Berechnung des neuen $d_{(l,k)}$, also die Wahrscheinlichkeit, mit der ein HMM_l (welches aus HMM_i und HMM_j entsteht) die Sequenz X_k emittiert, gestaltet sich wie folgt. Das HMM_l soll folgendermaßen konstruiert sein: Es besteht aus den Zuständen und den Transitionen beider HMM. Auch die Emissionen werden einfach übernommen. Es werden keine zusätzlichen Transitionen zwischen den HMM eingefügt bzw. die Übergangswahrscheinlichkeit zu Zuständen des anderen HMM beträgt 0. Die einzige Änderung betrifft die Startverteilung Π_l. Hatte ein Zustand s_t aus HMM_i oder HMM_j vorher die Startwahrscheinlichkeit π_t, so erhält er in Π_l den Wert $\pi_t/2$. Auf diese Weise startet HMM_l zu 50% Wahrscheinlichkeit in einem Zustand des „alten“ HMM_i und zu 50% Wahrscheinlichkeit in einem Zustand des „alten“ HMM_j. Danach verhalten sich die disjunkten Teile genauso wie die beiden ursprünglichen HMM. Die Startwahrscheinlichkeit Π_l ist somit in der Summe gleich 1. Wie im Modell zu sehen, wird nun solange „geclustert“ bis alle Klassen in einer einzigen Klasse vereinigt sind. Das dabei aufgebaute Newick-Format kann anschließend zur Konstruktion eines phylogenetischen Baums genutzt werden.

3.6.7 Implementierung und Tests

Der Javacode der Implementierung verwendet sowohl Teile des BioJava-Pakets (siehe Kapitel 2.6.2) als auch der HMM-Implementierung aus Kapitel 3.1. Aus dem BioJava-Paket werden die Konzepte der Alphabetstruktur verwendet. Den Baum-Welch- und Forward-Algorithmus stellt die `HMMToolbox` der HMM-Implementierung zur Verfügung (siehe Kapitel 2.4.2 und 3.1.4).

Die Implementierung wird von einer Klasse `PhyloBuilder` aufgerufen, welche sämtliche Arbeitsvorgänge nacheinander auslöst. Zunächst werden die Sequenzen einer vorliegenden FASTA-Datei mittels `Filereadfasta` ausgelesen. Die Längen der gewonnenen Sequenzen werden vom `HMMInitializer` verwendet, um unter anderem die Austrittwahrscheinlichkeiten der HMM in den Magical-State zu bestimmen. Diese Länge soll dem erwarteten Wert der Emissionslänge des HMM_i entsprechen. Also ist die Ausgangswahrscheinlichkeit (die Wahrscheinlichkeit das HMM in den Magical-State zu verlassen) $1/|X_i|$. Des Weiteren definiert der `HMMInitializer` ebenso die Eingangs-, Transitions- und Emissionswahrscheinlichkeiten für eine Sequenz X_i. Die Eingangswahrscheinlichkeiten für jedes HMM bestimmen sich aus der Zahl der gewählten Zustände des HMM. Diese enthält der Konstruktor des `HMMInitializer`. Nach Initialisierung der HMM wird für jedes HMM_i und die dazugehörige Sequenz X_i der Baum-Welch-Algorithmus aus der `HMMToolbox` zur Spezialisierung der HMM_i auf X_i aufgerufen. Die Distanzbestimmung der trainierten HMM erfolgt mittels Berechnung der Forward-Werte. Dies geschieht, indem für jedes HMM_i die Forward-Werte aller $X_1 ... X_n$ in ein 2-dimensionales Array geschrieben werden. Die nun gewonnene Distanzmatrix wird von der Klasse `Distancematrix` geclustert. Dabei werden rekursiv in jedem Iterationsschritt der Methode `Distancematrix .Cluster()` die beiden am nächsten liegenden HMM und Sequenzen ermittelt. Diese sind jene Werte, für die die Wahrscheinlichkeiten im Mittel am größten sind. Jene gefundenen Elemente fasst `Distancematrix.Cluster()` zu einem Objekt gemäß Gleichung 3.8 zusammen, welches einen gemeinsamen Vorfahren repräsentiert. Zu diesem Objekt wird eine neue Zeile und eine neue Spalte berechnet. Seien dies i und j mit $i < j$. Diese werden in die kleinere Zeile i und Spalte i der Sequenz überschrieben. Zur Zeile und Spalte mit dem höheren Index j rücken alle anderen Zeilen $> j$ und Spalten $> j$ auf, so dass eine um 1 kleinere Matrix entsteht. Die Rekursion ist endlich, da in jedem Aufruf sichergestellt ist, dass zwei Elemente zu einem Objekt zusammengefügt werden und das Abbruchkriterium eine Matrixgröße von 1×1 ist. Zuletzt wird das Newick-Format ausgegeben.

Für UPGMA werden Dateien im von ClustalW [THG94] verwendeten ALN-Format benötigt. Die Klasse `FilereadAlignment` ist in der Lage, jedes beliebige Multiple Sequenzalignment in diesem Format einzulesen. Der Hamming-Abstand der Sequenzen wird durch die Methode `Upgmadistance.compareStringDistances()` bestimmt, die diesen analog zur HMM-Methode in ein 2-dimensionales Array schreibt. Das Clustern funktioniert analog zur HMM-Methode, wobei nun die Minimalwerte zusammengefügt werden, und es handelt sich um eine symmetrische Matrix. Auch hier wird parallel das Newick-Format entwickelt und ausgegeben. Die Ergebnisse beider Algorithmen können nun betrachtet und bewertet werden.

3.6.8 Ergebnisse

Bei der Durchführung einer Testreihe kam es zu folgendem Ergebnis. Offenbar gruppiert der Algorithmus gleich lange Sequenzen mit vielen Unterschieden eher als Sequenzen, die bis auf einen kleinen Größenunterschied identisch sind. Zu sehen ist dies an folgender Reihe, bei der stets drei Sequenzen mit einer Ausgangslänge von 20 Basen als Eingabe dienen. Die im Ursprung gleichen Sequenzen wurden so verändert: Bei den ersten 5 der insgesamt 8 Testläufen werden in Sequenz X_0 alle „A“ in „T“ umgewandelt, so dass zu Sequenz X_1 7 Abweichungen auftreten. In Sequenz X_2 werden zusätzliche „C“ eingefügt, so dass sie länger ist als Sequenz X_1 (und X_0). Beobachtet wird nun, mit welcher anderen Sequenz die Vergleichssequenz Sequenz X_1 zusammengefasst wird (X_0 oder X_2). Die beobachteten Werte können Tabelle 3.9 entnommen werden.

Klar festzustellen ist, dass viel eher „nach Länge“ geclustert wird als nach ähnlichen Basen. Dies ist nicht wünschenswert, da z.B. das Einfügen nur einiger Buchstaben (wie es in der Natur geschehen kann) die Ähnlichkeitsbewertung übermäßig verschlechtert, so dass nicht verwandte Sequenzen zusammengefasst werden.

Diskussion

Die Bewertung phylogenetischer Bäume unterliegt zweierlei Problemen. Zum einen wurde die Performance der HMM auf Sequenzen überschätzt. Aufgrund der numerischen Einschränkungen der Datenstrukturen in Java konnten keine Sequenzen der ursprünglich geplanten Länge verarbeitet werden. Durch das stetige Aufmultiplizieren während der Forward-Berechnungen konvergieren die Wahrscheinlichkeitswerte zu 0. Bei Sequenzen von Längen um 200 führt dies zu nicht mehr darstellbaren Werten und schließlich dazu, dass keine Berechnungen durchgeführt werden können. Dies führt dazu, dass keine Aussage über Verwandschaftsbeziehungen real existierender Spezies gemacht werden kann. Deshalb konnten nur kürzere Sequenzen benutzt werden. Hierbei kam das zweite Problem zu tragen. Die Form eines phylogenetischen Baumes hängt im wesentlichen von der Distanzbewertung der Spezies untereinander ab. Es war nicht überraschend, dass zu großen Teilen Bäume entstanden, welche eine andere Phylogenie aufwiesen. Das Bewerten dieser Ergebnisse ist nur in einem größeren Zusammenhang möglich. Im direkten Vergleich zu UPGMA sei erwähnt, dass der Zeitaufwand für UPGMA mit einem naiv bewertetem Rechenaufwand von $O(n^2)$ für die geplanten Sequenzen im Millisekundenbereich liegt, während die HMM-Methode einige Sekunden benötigt, um aufgrund des oben genannten Problems ergebnislos zu bleiben. Der zusätzlich nötige Speicherbedarf zur Verwaltung der HMM (siehe Kapitel 3.1.5) ist ebenso negativ an zu merken. Für UPGMA wird lediglich ein String-Array sowie eine $(n \times n)$-Matrix verwaltet. Nach Durchführung einer Testreihe muss gesagt werden, dass das Clustering mit HMM (so wie es implementiert wurde) nicht zufriedenstellend ist.

Test	$d_{(X_0,X_1)}$	$d_{(X_2,X_1)}$	X_1 geclustert mit
1	7	6	X_0
2	7	3	X_0
3	7	2	X_0
4	7	1	X_2
5	7	0	X_2
6	14	2	X_0
7	14	1	X_2
8	14	0	X_2

Tabelle 3.9: Es wird angegeben, mit welcher der beiden Sequenzen X_0 oder X_2 die Sequenz X_1 zusammengefasst wird. Abweichungen von X_0 zu X_1 sind geänderte Basen (bei gleicher Länge), Abweichungen von X_2 zu X_1 sind zusätzlich eingefügte Basen, die Sequenz wird um diese Stellen länger. Zu sehen ist, dass eine Zunahme der Länge eher als Abweichung interpretiert wird als ausgetauschte Basen.

4 Zusammenfassung

Laut Diplomprüfungsordnung (DPO) Informatik ist die Projektgruppe eine Lehrveranstaltung, die sich über zwei Semester erstreckt und zwei Ziele verfolgt. Zum einen sollen „Methoden der Informatik angewendet“ und „Informatikkenntnisse vertieft“ [Unia] werden und der Studierende soll lernen, sich nach dem Studium die ständig fortschreitenden Methoden und Erkenntnisse der Informatik anzueignen und seinem Bedarf anzupassen. Zum anderen soll er befähigt werden, „ein konkretes Problem in Teilprobleme zu zerlegen, die Teilprobleme einer koordinierten Bearbeitung zuzuführen [und] Lösungen von Teilproblemen zu einer Gesamtlösung zu integrieren“ [Unib]. Die Projektgruppe fasst Lehrinhalte eines Seminars, einer Spezialvorlesung, eines Praktikums und eines Kolloquiums zusammen.

Die konkrete Umsetzung dieser Vorgaben in der PG487 begann mit der Anfertigung von Präsentationen zu Themen, die in das Arbeitsfeld der PG einführen sollten und innerhalb eines zweitägigen Kompaktseminars in Bommerholz vorgestellt wurden. Neben diesen Vorträgen wurden Tutorien zu bestimmten Softwaretools durchgeführt, die eine Einführung in die für die Koordination der einzelnen Arbeiten notwendigen Werkzeuge darstellten. Im Anschluss an die Seminarphase fand zweimal wöchentlich ein Treffen statt, um mit der ganzen Gruppe Ziele zu setzen, Aufgaben zu diskutieren und Ergebnisse vorzustellen. Die Leitung dieser Sitzungen, sowie das Verfassen eines Protokolls, oblag jeweils einem PG-Teilnehmer, so dass sich jeder mehrfach darin üben konnte. Im Anschluss an die Sitzungen gab es seitens der Gruppe eine Kritik zur Sitzungsleitung und ggf. zu gehaltenen Vorträgen, sodass man seine eigene Leistung einschätzen konnte. Zudem konnte durch die regelmäßige, gegenseitige Kritik ein produktives und weitgehend spannungsfreies Arbeitsklima geschaffen werden.

Um zunächst ein gemeinsames Grundverständnis für die Bioinformatik zu erarbeiten, wurden Kleingruppen gebildet, die sich mit dem Lokalen bzw. Globalen Alignmentproblem, dem Dekodierungs- und Trainingsproblem für HMM und anderen, grundlegenden Problemen und Lösungsverfahren auseinandersetzten. Im Anschluss daran wurden zwei Gruppen gebildet, die sich mit der Bearbeitung größerer Aufgaben beschäftigten. Sechs der PG-Teilnehmer entwickelten einen Evolutionären Algorithmus zur Berechnung eines Multiplen Sequenzalignment (siehe Kapitel 3.2), die anderen einen EA zum Optimieren von HMM, die ein solches Alignment erstellen (siehe Kapitel 3.3). Zu Beginn des zweiten Semesters der PG wurden Vorschläge zu Themengebieten, die in diesem Semester bearbeitet werden sollten gesammelt. Für diese Themen wurden dann, gemäß vorher geäußerter Präferenzen, Gruppen zu je zwei bis drei Personen gebildet. Diese Kleingruppen organisierten sich selbstständig und agierten weitgehend unabhängig voneinander, einzig die Untergruppe, welche sich um die grundlegende Struktur der HMM kümmerte, arbeitete in Absprache mit den anderen Gruppen. Zur Koordination wurden ein Programm

zur Versionierung, ein WIKI sowie die wöchentlichen Treffen genutzt. Die Präsentation der Ergebnisse der Arbeiten fand ebenfalls im Rahmen der gemeinsamen Treffen statt.

Im Folgenden wird ein Überblick über die Arbeit der fünf Gruppen des zweiten Semesters gegeben. Als erstes ist die Gruppe, die sich mit der Implementierung der HMM beschäftigte, zu nennen (siehe Kapitel 3.1). Da die Implementierung von allen anderen Gruppen benutzt werden sollte, waren ihre Ergebnisse sowohl für den zeitlichen Ablauf, als auch für die tatsächlichen Ergebnisse entscheidend. Der Speicherplatzbedarf der HMM konnte gegenüber der Implementierung in BioJava deutlich gesenkt und die Handhabbarkeit verbessert werden, sodass sie einfacher auf verschiedene Probleme angepasst werden konnten. Das Ziel innerhalb weniger Wochen eine abgeschlossene Implementierung vorweisen und die Mitglieder der Gruppe in die übrigen Gruppen aufteilen zu können musste jedoch zugunsten einer umfassenderen und verlässlicheren Implementierung aufgegeben werden. Die Gruppe, die sich mit Multiplen Sequenzalignments durch Evolutionären Algorithmen auf HMM beschäftigte (siehe Kapitel 3.3), war die Fortführung der entsprechenden Gruppe des ersten Semesters. Sie verwendete Profile HMM als eine Spezialisierung von HMM. Zur Optimierung mittels eines EA wurden eigene Operatoren entwickelt und die Güte der Ergebnisse in einer Testumgebung ermittelt. Die Gruppe zur Genvorhersage beschäftigte sich mit dem Finden von Genen in DNA-Sequenzen (siehe Kapitel 3.4). Dazu wurde ein spezielles HMM konzipiert sowie ein eigenes Verfahren zum Bestimmen optimaler Pfade implementiert. Eine Reihe weiterer eigener Methoden wurde erarbeitet, eine Evolutionsstrategie zur Optimierung angelegt und das Verfahren wurde am Beispiel eines Mensch-Maus-Datensatzes getestet. Im Teilprojekt für die Strukturvorhersage von Proteinen war das Ziel, die Sekundärstruktur von Proteinen allein anhand ihrer gegebenen Aminosäuresequenz vorherzusagen (siehe Kapitel 3.5). Dazu wurden auch hier HMM verwendet, die mit speziellen Operatoren in einer ES optimiert wurden. Die Güte der Ergebnisse wurde ermittelt und mit dem aktuellen Stand der Forschung verglichen. In der Gruppe zur Berechnung Phylogenetischer Bäume durch Clustering wurden HMM zum Erstellen eines Clustering von verwandten Sequenzen benutzt (siehe Kapitel 3.6.1). Die Ergebnisse wurden auf ihre Aussagekraft überprüft und mit bekannten Verfahren verglichen.

Neben den fachlich erreichten Zielen soll die allgemeine Arbeitsweise und Organisation betrachtet werden. In der PG487 hat sich gezeigt, dass eine konkrete Problemstellung als Einstieg in eine Projektgruppe und einen größeren Themenbereich besser gewesen wäre. Nach der Einarbeitung in dieses Thema, die gleichzeitig eine Orientierung bietet, könnte entschieden werden, ob weitere Seminarvorträge notwendig sind. Weitere Ziele könnten danach von der Gruppe bestimmt werden und der Arbeitsaufwand sowie die Fähigkeiten der anderen Mitglieder aus den gewonnenen Erfahrungen besser eingeschätzt werden. Ein enger gegenseitiger Bezug der Themen ist wünschenswert, die je nach Arbeitsaufwand mindestens zu dritt bearbeitet werden, damit es zu einer echten Gruppenarbeit kommt und auch ambitioniert Ziele erreicht werden können. Die notwendige Software sollte vorhanden sein und es sollte nach Möglichkeit auf (frei) verfügbare Implementierungen zurück gegriffen werden, um Verzögerungen zu vermeiden. Weiterführende Ziele sollten noch vor den Semesterferien definiert werden, damit diese Zeit zur Einarbeitung genutzt werden kann. So kann mit dem Beginn des zweiten Semesters das Projekt zügig

fortgeführt werden.

Zur Anwendung und Vertiefung von Methoden der Informatik gehörte der Umgang mit wichtigen Software-Werkzeugen wie beispielsweise dem Programm zur Versionierung, dem Batch-System des LS11, dem SPOT-Paket oder auch LaTeX. Neben der Programmierung war das Erstellen von Dokumenten von großer Bedeutung. Zum einen war das Erstellen eines zusammenhängenden Dokuments wie des Zwischenberichts oder dieses Endberichts durch eine Gruppe von zwölf Personen in Hinblick auf Einheitlichkeit, Organisation sowie der Vermeidung von Redundanz eine anspruchsvolle Aufgabe. Zum anderen musste der einzelne auf eine präzise Ausdrucksweise, einen sinnvollen Textaufbau und dem korrekten Umgang mit Quellen achten, sodass diese Arbeit auch eine praktische Übung für das Schreiben einer Diplomarbeit ist.

Die für eine erfolgreiche berufliche Karriere geforderten „Soft-Skills“ werden im Gegensatz zur Projektgruppe im Rahmen der sonst üblichen Lehrveranstaltungen einer Universität kaum trainiert. Durch das Leiten der Sitzungen konnten Moderation und das Führen einer Gruppe geübt werden. Das Halten von Vorträgen, in denen über Fortschritte der eigenen Arbeit informiert wurde, war nicht nur ein Training zur Rhetorik und Medienverwendung, sondern auch eine Gelegenheit, andere von eigenen Ideen zu überzeugen. Durch die lange Dauer der Lehrveranstaltung von einem Jahr war eine selbstständige Organisation der Gruppen und eine gute Zeitplanung notwendig. Zusammen mit der praktischen Anwendung des im Studium erworbenen, theoretischen Wissens ergibt sich ein konkreter Bezug zur Arbeit im späteren Berufsleben. Abschließend kann festgestellt werden, dass die zu Beginn dieses Kapitels genannten, allgemeinen Ziele einer Projektgruppe erreicht wurden und eine allgemeine Eignung von Methoden der Computational Intelligence auf Probleme der Bioinformatik nachgewiesen wurde. Das Potenzial der in den einzelnen Gruppen erarbeiteten Ansätze ist vermutlich noch lange nicht ausgeschöpft und bietet wie das gesamte Feld der Bioinformatik viel Spielraum für weitere Entwicklungen.

Literaturverzeichnis

[AGMM90] Altschul, S.-F. ; Gish, W. ; Miller, W. ; Myers, D.-J. E.-W. und L. E.-W. und Lipman: Basic local alignment search tool. In: *Journal of Molecular Biology* 215 (1990), October, Nr. 3, S. 403–410

[Apa99] Apache Software Foundation: *Xerces*. `http://xerces.apache.org`. Version: 1999. – http://xerces.apache.org

[BB06] Bartz-Beielstein, T.: *Experimental research in evolutionary computation. The new experimentalism*. Springer-Verlag GmbH, 2006. – 214 S. – ISBN 3540320261

[BBL06] Bartz-Beielstein, T. ; Lasarczyk, M. C. und P. C. und Preuß: Sequential parameter optimization toolbox / Universität Dortmund, Germany. 2006 (CI-15x/06). – Forschungsbericht

[Bel57] Bellman, R.: *Dynamic programming*. Princeton University Press, 1957

[Ber06] Berglund, Anders: *Extensible Stylesheet Language (XSL) Version 1.1*. W3C Recommendation `http://www.w3.org/TR/xsl11/`, 2006

[BG98] Beck, K. ; Gamma, R.: *JUnit*. `http://www.junit.org`. Version: 1998. – http://www.junit.org

[BHK97] Barrett, C. ; Hughey, R. ; Karplus, K.: Scoring hidden markov models. In: *CABIOS* 13 (1997), Nr. 13(2), S. 191–199

[BPSM+04] Bray, T. ; Paoli, J. ; Sperberg-McQueen, C. M. ; Maler, E. ; Yergeau, F.: *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C Recommendation `http://www.w3.org/TR/REC-xml`, September 2004

[BS02] Beyer, H.-G. ; Schwefel, H.-P: *Evolution strategies - a comprehensive introduction*. Kluwer academic publishers, 2002. – 3–52 S.

[CWS01] Cawley, S.-E. ; Wirth, A.-I. ; Speed, T.-P.: Phat–a gene finding program for plasmodium falciparum. In: *Mol. Biochem. Parasitol.* 181 (2001), S. 167–174

[DEKM99] Durbin, R. ; Eddy, S.-R. ; Krogh, A. ; Mitchison, G.: *Biological sequence analysis : Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1999. – ISBN 0521629713

[Dir07] DIRECT, Biology: *Table 2.* `http://www.biology-direct.com/content/1/1/10/table/T2`. Version: 2007. – http://www.biology-direct.com/content/1/1/10/table/T2

[Edd95] EDDY, S.-R.: Multiple alignment using hidden markov models. In: *Proc Int Conf Intell Syst Mol Biol* 3 (1995), S. 114–120

[Edd98] EDDY, S.-R.: Profile hidden markov models (review). In: *Bioinformatics* 14 (1998), Nr. 9, S. 755–763

[Fin03] FINK, Gernot A.: *Mustererkennung mit Markov-Modellen.* Stuttgart – Leipzig – Wiesbaden : B. G. Teubner, 2003 (Leitfäden der Informatik)

[Fog62] FOGEL, L.-J.: Autonomous automata. In: *Industrial Research* 4 (1962), S. 14–19

[HD06] HÜTT, M.-T. ; DEHNERT, M.: *Methoden der Bioinformatik.* Springer Berlin Heidelberg, New York, 2006

[Hol62] HOLLAND, J.-H.: Outline for a logical theory of adaptive systems. In: *JACM* 9 (1962), S. 297–314

[KBSM+94] KROGH, A. ; BROWN, M. ; SAIRA MIAN, I. ; SJÖLANDER, K. ; HAUSSLER, H.-J.: Hidden markov models in computational biology : Applications to protein modeling / University of California, Santa Cruz. 1994 (235). – Forschungsbericht. – 1501–1531 S.

[Kem05] KEMPKA, M.: *Coverlipse.* `http://coverlipse.sourceforge.net`. Version: 2005. – http://coverlipse.sourceforge.net

[KS05] KABSCH, W ; SANDER, C: *DSSP.* `ftp://ftp.cmbi.ru.nl/pub/molbio/data/dssp`. Version: 2005. – ftp://ftp.cmbi.ru.nl/pub/molbio/data/dssp

[Law89] LAWRENCE, R.-R.: A tutorial on hidden markov models and selected applications in speech recognition. In: *Proceedings of the IEEE* 77 (1989), Nr. 2, S. 257–286

[Leo04] LEOPOLD, S.: *An alignment graph based evolutionary algorithm for the multiple sequence alignment problem*, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Diplomarbeit, 2004

[Les03] LESK, A.: *Bioinformatik - Eine Einführung.* Spektrum, Akad. Verlag Heidelberg, Berlin, 2003

[Löf01] LÖFFLER, Georg: *Basiswissen Biochemie.* Springer-Verlag GmbH, 2001. – ISBN 354067389

[Mar06] MARKOV, A.-A.: An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains. In: *Science in Context* 19.4 (2006), S. 591–600

[Mat06] MATHWORKS: *MATLAB*. `http://www.mathworks.de/`. Version: 2006. – http://www.mathworks.de/

[MGR05] MARTIN, J. ; GIBRAT, J.-F. ; RODOLPHE, F.: Choosing the optimal hidden markov model for secondary-structure prediction. In: *IEEE Intelligent Systems* 20 (2005), Nr. 6, S. 19–25

[NCB07] NCBI: *National Center for Biotechnology Information*. `http://www.ncbi.nlm.nih.gov/`. Version: 2007. – http://www.ncbi.nlm.nih.gov/

[Nee70] NEEDLEMAN, C.-D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. In: *Journal of molecular biology* 48 (1970), S. 443–453

[Pac02] PACHTER, L. et a.: Applications of generalized pair hidden markov models to alignment and gene finding problems. In: *Journal of Computational Biology* 9 (2002), S. 389–399

[PD98] POCOCK, M. ; DOWN, T.: *Biojava*. `www.biojava.org`. Version: 1998. – www.biojava.org

[PD06] POCOCK, M. ; DOWN, T.: *API 1.4 of BioJava software*. `http://www.biojava.org/docs/api14/index.html`. Version: 2006. – http://www.biojava.org/docs/api14/index.html

[Rec65] RECHENBERG, I.: *Cybernetic solution path of an experimental problem*. 1965

[RHJ99] RAGGETT, Dave ; HORS, Arnaud L. ; JACOBS, Ian: *HTML 4.01 Specification*. W3C Recommendation, `http://www.w3.org/TR/html401/`. `http://www.w3.org/TR/html401/`. Version: December 1999

[Sch65] SCHWEFEL, H.-P.: *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*, Technische Universität Berlin, Hermann Föttinger-Institut für Strömungstechnik, Diplomarbeit, März 1965

[SGD95] SEAN, R.-E. ; GRAEME, J.-M. ; DURBIN, R.: Maximum discrimination hidden markov models of sequence consensus. In: *Journal of Computational Biology* 2 (1995), Nr. 1, S. 9–23

[SKS+03] SCHWARTZ, S. ; KENT, W.-J. ; SMIT, A. ; ZHANG, Z. ; BAERTSCH, R. ; HARDISON, R. ; HAUSSLER, D. ; MILLER, W.: Human-mouse alignments with BLASTZ. In: *Genome Research* 13 (2003), Nr. 1, S. 103–107

[SM58] SOKALL, R.-R. ; MICHENER, C.-D.: A statistical method for evaluating systematic relationships. In: *University of Kansas Science Bulletin* 38 (1958), S. 1409–1438

[Sou00] South African National Bioinformatics Institute: *BLAST tutorial*. 2000. – http://www.za.embnet.org/mrc/tdr2004/Presentation/rudi/

[Sun94] Sun Microsystems: *Javadoc*. `http://java.sun.com/j2se/javadoc`. Version: 1994. – http://java.sun.com/j2se/javadoc

[SW81] Smith, T.-F. ; Waterman, M.-S.: Identification of common molecular subsequences. In: *Journal of Molecular Biology* 147 (1981), S. 195–197

[SWMW89] Sacks, J. ; Welch, W.-J. ; Mitchel, T.-J. ; Wynn, H.-P.: Design and analysis of computer experiments. In: *Statistical Science* 4 (1989), S. 409–423

[SWN03] Santner, T.-J. ; Williams, B. ; Notz, W.: *The design and analysis of computer experiments*. Springer-Verlag, 2003. – 283 S.

[THG94] Thompson, J.-D. ; Higgins, D. ; Gibson, T.-J.: ClustalW: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. In: *Nucleic Acids Res.* 22 (1994), 4673–4680. `http://www.cf.ac.uk/biosi/research/biosoft/Downloads/clustalw.html`. – http://www.cf.ac.uk/biosi/research/biosoft/Downloads/clustalw.html

[Unia] UniDo, Informatikdekanat: *Diplomprüfungsordnung Kerninformatik 1996*. `http://dekanat.cs.uni-dortmund.de/Ordnungen/DPO/DPOKI.html`. – http://dekanat.cs.uni-dortmund.de/Ordnungen/DPO/DPOKI.html

[Unib] UniDo, Informatikdekanat: *PG-Ordnung (Informationen der Projektgruppenbeauftragten der Universität Dortmund)*. `http://ls4-www.cs.uni-dortmund.de/PGB/alles/node4.html`. – http://ls4-www.cs.uni-dortmund.de/PGB/alles/node4.html

[Vit67] Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum. In: *Information Theory, IEEE Transactions on* 13 (1967), April, 260–269. `http://www.essrl.wustl.edu/~jao/itrg/viterbi.pdf`

[Weg05] Wegener, I.: *Skript zur Vorlesung Datenstrukturen, Algorithmen und Programmierung 2*. `http://ls2-www.cs.uni-dortmund.de/lehre/sommer2005/dap2/skript.pdf`. Version: 2005. – http://ls2-www.cs.uni-dortmund.de/lehre/sommer2005/dap2/skript.pdf

[Wei02] Weicker, K.: *Evolutionäre Algorithmen*. Teubner, Stuttgart, 2002

[WHPBK05] Won, K.-J. ; Hamelryck, T. ; Prügel-Bennett, A. ; Krogh, A.: Evolving hidden markov model for protein secondary structure prediction. In: *Evolutionary Computation* 1 (2005), Nr. 1, S. 33–40