UNIVERSITÄT DORTMUND

Naturinspirierte Verfahren zur Anpassung von Hidden Markov Modellen in der Bioinformatik

Martina Vaupel

Algorithm Engineering Report
TR07-2-007
Juli 2007
ISSN 1864-4503

Universität Dortmund
Fachbereich Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

Diplomarbeit

Naturinspirierte Verfahren zur Anpassung von Hidden Markov Modellen in der Bioinformatik

Martina Vaupel

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

26. März 2007

Gutachter:

Prof. Dr. Günter Rudolph
Prof. Dr. Thomas Bartz-Beielstein

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der Natur und im täglichen Leben begegnen uns immer wieder Prozesse, bei denen wir eine Ausgabe beobachten können. Diese kann die unterschiedlichsten Formen annehmen. Meist liegt diesen Beobachtungen bzw. diesen beobachteten Ausgaben ein bestimmter Prozess zugrunde, der die Ausgabe hervorruft. Um nun diesen Prozess besser verstehen zu können, obwohl nur die von ihm verursachten Ausgaben bekannt sind, wird dieser häufig durch ein stochastisches Modell nachgebildet. Liegt ein solches Modell vor, kann es letztendlich auch zur Vorhersage des internen Verhalten des Prozesses für beliebige Ausgaben verwendet werden. Dies ist beispielsweise dann interessant, wenn aus dem internen Verhalten Rückschlüsse auf Daten gezogen werden können, deren Beschaffung sonst nicht oder nur schwer möglich ist oder deren Beschaffung sehr kosten- oder zeitintensiv wäre.

Ein Hidden Markov Modell (kurz: HMM) ist eine Form eines solchen Modells und ist in der Lage, die Ausgabe der unterschiedlichsten Symbolsequenzen zu modellieren. Ihre Parameter werden so auf die jeweilige Problemstellung angepasst, dass sie die Wahrscheinlichkeit für die beobachteten Sequenzen von Ausgabesymbolen maximieren. Seit Anfang der siebziger Jahre werden sie erfolgreich in der Spracherkennung eingesetzt und 1992 schlug David Haussler sie erstmals für Problemstellungen aus dem Bereich der Bioinformatik vor. Mit ihrer Hilfe werden beispielsweise besondere Informationen aus DNA-Sequenzen extrahiert. In dieser Arbeit werden sie zur Vorhersage von sogenannten CpG-Inseln verwendet. Dies sind Teile der DNA, in denen das Dinukleotid CpG vermehrt vorkommt. CpG-Inseln sind beispielsweise für die Forschung nach den Ursachen der Entstehung von Krebserkrankungen überaus bedeutsam. Dies wird im ersten Teil von Kapitel 2 ausführlicher beschrieben.

Ein HMM besteht aus einer Menge von Zuständen, die durchlaufen werden. In jedem Zustand wird zunächst entsprechend einer Wahrscheinlichkeitsverteilung ein Symbol aus dem vorgegebenen Ausgabealphabet ausgegeben und anschließend, einer zweiten Wahrscheinlichkeitsverteilung folgend, ein Nachfolgezustand erreicht. Die Bezeichnung der Hidden Markov Modelle stammt daher, dass im Allgemeinen lediglich die Ausgabe-

sequenz bekannt ist, während die ihr zugrunde liegende Zustandssequenz unbekannt bzw. „versteckt“ ist.

Um ein HMM an eine Problemstellung anzupassen, müssen nun einerseits seine Topologie — also die Anzahl der Zustände — und andererseits die einzelnen Wahrscheinlichkeitsverteilungen angemessen gewählt werden. Zur Bestimmung der Wahrscheinlichkeitsverteilungen wird häufig der sogenannte Baum-Welch-Algorithmus verwendet. Dieser benutzt ein Gradienten-Abstiegs-Verfahren, um die Wahrscheinlichkeiten iterativ an eine Menge von vorgegebenen Trainingsbeispielen anzupassen. Dabei konvergiert der Algorithmus allerdings meist — abhängig von der Initialisierung der Parameter — in einem lokalen, jedoch nicht globalen Optimum.

Außerdem muss, bevor der Baum-Welch-Algorithmus angewendet werden kann, zunächst die Topologie des HMMs festgelegt werden. Dazu wird jedoch a priori Wissen über die Problemstellung benötigt, das in vielen Fällen nicht vorhanden ist. Ein weiterer Nachteil des Baum-Welch-Algorithmus ist, dass damit nur die Wahrscheinlichkeiten der gegebenen Sequenzen maximiert werden können, nicht aber die Wahrscheinlichkeit für eine korrekte Vorhersage der Informationen, die das spezielle Problem verlangt, in diesem Fall also die Wahrscheinlichkeit der korrekten Vorhersage von CpG-Inseln. HMMs und die Algorithmen dafür werden im zweiten Teil von Kapitel 2 näher beleuchtet.

Aufgrund der erwähnten Probleme werden häufig evolutionäre Verfahren als Alternative und Erweiterung des Baum-Welch-Algorithmus eingesetzt. Sie bieten zusätzlich den Vorteil, dass mit ihnen nicht nur die Wahrscheinlichkeiten selbst, sondern auch die Topologie des HMMs direkt verändert werden kann. Dies kann beispielsweise in Thomsen (2002) nachgelesen werden. Des Weiteren besteht die Möglichkeit, mit ihnen Profil-HMMs anzupassen, in denen schon bestehende Informationen über die Problemstellung in das Modell mit einbezogen werden, siehe Won u. a. (2005). Bisher wurden evolutionäre Algorithmen allerdings nicht systematisch angewendet, sondern dessen Parameter durch Experimente für jedes Problem separat ermittelt.

Im Kapitel 3 wird zunächst die Entwicklung der durch den Baum-Welch-Algorithmus generierten Lösungen über die Zeit beobachtet. Zusätzlich werden dort Gütemaße für HMMs eingeführt, die die Möglichkeit bieten sollen, verschiedene Lösungen zu beurteilen und miteinander zu vergleichen. Diese Gütemaße werden dann für die vom Baum-Welch-Algorithmus errechneten Lösungen bestimmt.

Kapitel 4 beschreibt die schrittweise Entwicklung verschiedener evolutionärer Ansätze. Mit Hilfe eines Verfahrens zur Anpassung der Parameter eines evolutionären Algorithmus (EA) werden für die verschiedenen Ansätze gute Einstellungen der Parameter, beispielsweise die Populationsgröße, ermittelt. Auch die Lösungen der verschiedenen evolutionären Ansätze werden durch die bereits erwähnten Gütemaße bewertet, um sie sowohl miteinander als auch mit den Ergebnissen des Baum-Welch-Algorithmus

vergleichen zu können. Außerdem werden verschiedene Aspekte der EAs untersucht, beispielsweise die Auswirkungen von Veränderungen der Parametereinstellungen auf die erzielte Güte.

Anschließend wird in Kapitel 5 die Möglichkeit eines hybriden Algorithmus, der den klassischen Ansatz in Form des Baum-Welch-Algorithmus mit den evolutionären Algorithmen vereint, untersucht. Dabei ist es besonders interessant, ob sich durch die Kombination der beiden Verfahren bessere Ergebnisse erzielen lassen als mit den einzelnen Verfahren. Dies wird ebenfalls anhand der Gütemaße beurteilt.

Die Ergebnisse werden in Kapitel 6 schließlich noch einmal zusammengefasst und es wird ein Ausblick auf zusätzliche Aspekte gegeben, die das in dieser Arbeit geschaffene Potential weiter ausschöpfen können und so für zukünftige Betrachtungen interessant wären.

Kapitel 2

Grundlagen

2.1 Biologische Grundlagen

Im folgenden Abschnitt wird ein Einblick gegeben, worum genau es sich bei CpG-Inseln handelt und woran sie erkannt werden können. Ihre Bedeutung für die Biologie wird anhand von zwei Anwendungsbeispielen dargestellt.

2.1.1 CpG Inseln

Die DNA-Sequenz des Menschen besteht aus den vier verschiedenen Nukleotiden Adenin, Cytosin, Guanin und Thymin, die kurz mit den Buchstaben A, C, G und T bezeichnet werden. Teile der DNA-Sequenz, in denen das Dinukleotid CG (im Folgenden als CpG bezeichnet) öfter vorkommt als im übrigen Teil, werden als CpG-Inseln bezeichnet.

Kommt das Nukleotid C in einem CpG Dinukleotid vor, wird es häufig methyliert. Dies bezeichnet den Prozess, bei dem ein H-Atom durch das Molekül CH_3 ersetzt wird. Solche Methyl-Cytosin Nukleotide mutieren mit hoher Wahrscheinlichkeit in Thymin, also T, sodass CpG deutlich seltener als erwartet auftritt. In manchen Abschnitten der Gene wird diese Methylierung allerdings unterdrückt, sodass wesentlich mehr CpG Dinukleotide auftreten als in der restlichen DNA-Sequenz. Diese Bereiche kommen speziell an den Anfangsregionen von Genen vor, woraus sich auch ihre besondere Bedeutung in der Biologie ergibt. Im Folgenden werden als Beispiele die Krebsentstehung und die Epigenetik näher betrachtet.

Die Umwandlung von Methyl-Cytosin (C) in Thymin ist eine Möglichkeit der Deaminierung und erfolgt ohne äußere Einwirkung. Eine Deaminierung tritt beispielsweise auch auf, wenn Adenin (A) spontan in sogenanntes Hypoxanthin umgewandelt wird. Außer der erstgenannten Möglichkeit ist allen anderen Deaminierungen jedoch eines gemein: Ihre Produkte sind normalerweise nicht in der DNA enthalten und können daher von Reparaturenzymen entdeckt und beseitigt werden. Dies ist bei der Deaminierung

von Methyl-Cytosin in Thymin nicht möglich, da Thymin ein natürlicher Baustein der DNA ist. Dadurch entstehen von Reparaturenzymen unentdeckte Veränderungen in der DNA, die — je nach dem, wo sie auftreten — schwerwiegende Folgen haben können.

In Genen bilden immer jeweils drei aufeinander folgende Nukleotide ein sogenanntes Codon. Bei der Übersetzung von Genen in Proteine steht ein solches Codon wiederum für eine bestimmte Aminosäure. Eines dieser Codons, nämlich ATG, das die Aminosäure Methionin codiert, fungiert als Startsignal. Andererseits fungiern drei verschiedene Codons (TAG, TAA und TGA) als Stopsignale und beschreiben keine Aminosäure. Eine Folge von Codons, also auch Aminosäuren, zwischen einem Start- und einem Stopsignal bilden ein bestimmtes Protein. Tritt eine solche unentdeckte Veränderung (Mutation) durch die Deaminierung von Methyl-Cytosin innerhalb eines Gens auf, kann sie Auswirkungen auf die Erzeugung von Proteinen zur Folge haben.

Durch diese Mutationen werden unter Umständen andere Aminosäuren erzeugt als sie für das Protein erforderlich wären. Außerdem besteht die Möglichkeit, dass zu früh ein Stopsignal erfolgt, beispielsweise, weil CAG in TAG deaminiert wurde. So können verkürzte, inaktive Formen von Proteinen entstehen, die nicht in der Lage sind, ihre Aufgaben im Körper zu erfüllen. Geschieht dies in den Onkogenen oder den Tumorsuppressoren, also bestimmten Klassen von Proteinen, kann so eine Disposition für eine Krebserkrankung entstehen.

In diesem Zusammenhang ist die Tatsache von besonderem Interesse, wie es dazu kommt, dass sich in CpG-Inseln zwar Methyl-Cytosin befindet, dieses allerdings nicht in Tymin deaminiert wird. Somit kommt es nicht zu den Mutationen und den damit verbundenen Folgen. Für die Forschung ist es wichtig, CpG-Inseln zu identifizieren, um dieses besondere Verhalten untersuchen zu können.

Ein weiteres Gebiet, für das CpG-Inseln interessant sind, ist die Epigenetik. Dabei geht es darum, wie die Genaktivität reguliert werden kann. Veränderungen an den durch die DNA beschriebenen Proteinen durch die Mutation von Methyl-Cytosin in Thymin verursachen wiederum Veränderungen an der lokalen DNA-Struktur. Dies führt dazu, dass einzelne Gene oder Bereiche der Gene nicht mehr in Proteine übersetzt werden können. Gene, die nicht mehr übersetzt werden können, werden als inaktiv bezeichnet.

Hieraus ergeben sich mögliche Anwendungen der Identifikation von CpG-Inseln. Beispielsweise könnten durch die gezielte Methylierung der Cytosine innerhalb von CpG-Inseln mutierte Gene stillgelegt werden. Dadurch könnte eine Krebszelle zu einer normalen Zelle werden. Eine weitere denkbare Anwendung wäre, das Protein, das den programmierten Zelltod (Apotose) unterdrückt, zu inaktivieren. Dies würde zu einem Absterben der Krebszellen führen. Auf diesem Gebiet wird in der Biologie noch viel geforscht. Dabei führt das genauere Verständnis solcher Prozesse immer wieder zu neuen Anwendungsmöglichkeiten.

Die bei den folgenden Untersuchungen zum Training verwendeten Beispieldaten für CpG-Inseln sind die gleichen, die auch in Durbin u. a. (1998) verwendet wurden und stehen unter der URL `ftp://ftp.ebi.ac.uk/pub/databases/cpgisle/` zum Download zur Verfügung.

2.2 Algorithmische Grundlagen

Die Vorhersage von CpG-Inseln in der DNA-Sequenz entspricht der Ermittlung einer Markierung für jedes Nukleotid in der Sequenz. Diese gibt an, ob sich das jeweilige Nukleotid innerhalb oder außerhalb einer CpG-Insel befindet. Um diese Markierungssequenz zu bestimmen, werden hier Hidden Markov Modelle verwendet. Im Folgenden werden diese vorgestellt und formal definiert. Um ein Hidden Markov Modell an eine Problemstellung anzupassen, wird häufig der Baum-Welch-Algorithmus eingesetzt. Der anschließend präsentierte Viterbi-Algorithmus bildet die Grundlage, um zu einer gegebenen DNA-Sequenz die CpG-Inseln vorherzusagen. Am Ende dieses Kapitels folgt eine kurze Beschreibung von evolutionären Algorithmen und ihrer Funktionsweise. Hier werden sie alternativ zum Baum-Welch-Algorithmus verwendet, um die Parameter eines Hidden Markov Modells anzupassen.

2.2.1 Hidden Markov Modelle

Hidden Markov Modelle gehen auf den russischen Mathematiker Andrej Andreevič Markov (*1856 – †1922) zurück. Vielen Anwendungen liegt ein stochastischer Prozess zugrunde. Häufig besteht jedoch kein direkter Zugriff auf den Prozess, sondern es sind lediglich verschiedene Ausgaben beobachtbar. Die zugehörigen Zustände sind verborgen. HMMs stellen eine Möglichkeit dar, derartige Prozesse zu modellieren. In dem vorliegenden Fall ist die beobachtbare Ausgabe die Nukleotidsequenz. Aktuelle Forschungen auf dem Gebiet der Bioinformatik gehen davon aus, dass die zugrunde liegende Zustandssequenz Schlussfolgerungen über die Bedeutung der DNA-Sequenz zulässt. Formal lassen sich HMMs wie folgt beschreiben.

Definition 1. *Ein Hidden Markov Modell wird durch das Tupel* $\Theta := (\mathcal{S}, \Gamma, A, E)$ *der Modellparameter beschrieben mit*

- *Zustandsmenge* $\mathcal{S} = \{s_0, s_1, \ldots, s_n\}$ *mit einem festen Startzustand* s_0*,*
- *Ausgabealphabet* $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_m\}$*,*
- $(n+1) \times n$ *Matrix* A *der Übergangswahrscheinlichkeiten;* $a_{s,s'}$ *bezeichnet dabei die Wahrscheinlichkeit von einem Zustand* $s \in \mathcal{S}$ *in den Zustand* $s' \in \mathcal{S} \backslash \{s_0\}$ *zu gelangen,*

- *$n \times m$ Matrix E der Ausgabewahrscheinlichkeiten; $e_s(\gamma)$ beziffert die Wahrscheinlichkeit im Zustand $s \in \mathcal{S} \backslash \{s_0\}$ ein bestimmtes Symbol $\gamma \in \Gamma$ zu emittieren. Im Startzustand wird kein Symbol ausgegeben.*

In der Literatur werden teilweise zusätzlich die Wahrscheinlichkeiten, in einem bestimmten Zustand zu starten, verwaltet. Dies erübrigt sich durch die Einführung des Startzustandes s_0, in dem kein Symbol ausgegeben wird. Es wird demnach immer in dem Startzustand gestartet und die Wahrscheinlichkeiten $a_{s_0,s}$, von s_0 aus in einen beliebigen anderen Zustand s zu gelangen, entsprechen in Modellen, in denen kein Startzustand verwaltet wird, den Wahrscheinlichkeiten in diesem Zustand s zu starten. Dies ermöglicht eine elegantere Handhabung. Dabei muss allerdings gewährleistet werden, dass es nicht möglich ist, von einem anderen Zustand zurück in den Startzustand zu gelangen. Dies geschieht hier durch die Gestaltung der Matrix A. Da es sich um eine $(n+1) \times n$ anstelle einer $(n+1) \times (n+1)$ Matrix handelt, ist eine Rückkehr in s_0 durch die fehlenden Einträge für die Wahrscheinlichkeiten der Zustandsübergänge bereits ausgeschlossen. Ein einfaches Beispiel für ein HMM wird in Abbildung 2.1 dargestellt.

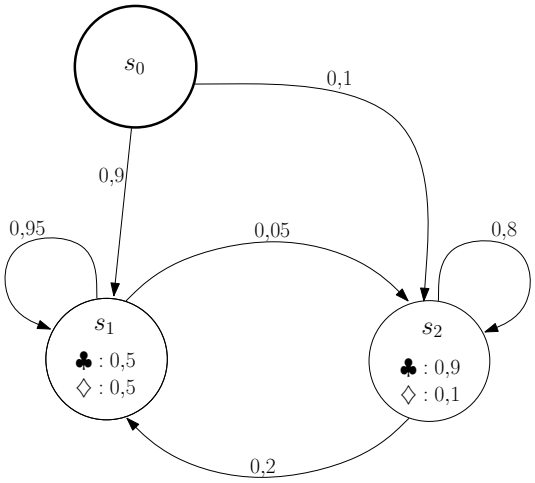


Abbildung 2.1: *Beispiel für ein HMM mit zwei Zuständen s_1 und s_2, einem Startzustand s_0 und zwei Ausgabesymbolen ♣ und ♢.* Die Übergänge von einem Zustand s in einen anderen Zustand s' sind mit dem Wert $a_{s,s'}$ markiert und jeder Zustand $s \neq s_0$ mit den Wahrscheinlichkeiten $e_s(\clubsuit)$ und $e_s(\diamondsuit)$, in ihm das jeweilige Symbol auszugeben.

Das wesentliche Problem bei der Arbeit mit HMMs besteht darin, die Modellparameter Θ festzulegen. Die Anzahl der Zustände und die möglichen Zustandsübergänge werden als Topologie des HMMs bezeichnet. Für die Topologieanpassung ist allerdings kein effizienter Algorithmus bekannt. Hierfür werden in den späteren Kapiteln die evolutionären Algorithmen verwendet. Zu einer gegebenen Topologie stellt der Baum-Welch-Algorithmus eine Möglichkeit dar, die Übergangs- und Ausgabewahrscheinlichkeiten an eine vorgegebene Menge von Ausgabesequenzen anzupassen.

2.2.2 Baum-Welch-Algorithmus

Die Elemente der vorgegebenen Menge von Ausgabesequenzen werden Trainingsbeispiele genannt und mit $x^1, x^2, \ldots, x^k$ bezeichnet. Jedes Trainingsbeispiel x^j besteht dabei aus einer Nukleotidsequenz $x_1^j x_2^j \ldots x_l^j \in \Gamma^l$. Sei $\text{Prob}(x^j|\Theta)$ die Wahrscheinlichkeit, dass das Trainingsbeispiel x^j von dem HMM mit den Modellparametern Θ ausgegeben wird. Das Ziel des Baum-Welch-Algorithmus ist es, die Wahrscheinlichkeit, dass alle Trainingsbeispiele von diesem HMM erzeugt werden, zu maximieren. Formal bedeutet dies die Maximierung von $\prod_{j=1}^{k} \text{Prob}(x^j|\Theta)$.

Da die zu den Trainingsbeispielen gehörenden Zustandssequenzen nicht bekannt sind, wird ein iterativer Ansatz zur Abschätzung der Modellparameter angewendet. Die verbreitetste Methode in diesem Zusammenhang stellt der Baum-Welch-Algorithmus dar. Der grundlegende Ablauf des Algorithmus während jeder Iteration ist der Folgende: Für die aktuellen Parameter Θ sei $A_{s,s'}$ die erwartete Anzahl an Zustandsübergängen von Zustand s nach s' während der Ausgabe der Trainingsbeispiele. Bezeichne $E_s(\gamma)$ die dabei erwartete Anzahl an Emissionen des Symbols γ im Zustand s. Diese beiden Werte werden berechnet und dienen als Grundlage für neue Modellparameter Θ. Basierend auf diesen neuen Modellparametern wird in der nächsten Iteration das Vorgehen wiederholt. Dieser Prozess konvergiert — abhängig von der Initialisierung der Modellparameter — in einem lokalen, meist aber nicht globalen Optimum. Das Vorgehen des Algorithmus wird nun detaillierter erläutert.

Eine Folge von durchlaufenen Zuständen heißt Pfad. Sei $\Pi^j = \pi_0^j, \pi_1^j, \ldots \pi_n^j$ der während der Emission der Sequenz x^j durchlaufene Pfad. Dabei bezeichnet π_i^j den Zustand, in dem sich das Modell während der Ausgabe des i-ten Symbols der Sequenz befindet. Da nur im Startzustand gestartet werden kann, gilt $\pi_0^j = s_0$. Die erwartete Anzahl $A_{s,s'}^j$ eines Zustandsübergangs von s nach s' in dem Trainingsbeispiel x^j läßt sich durch die Addition über alle Positionen des Pfades Π^j, an denen dieser Zustandsübergang vorkommen könnte, bestimmen. Es gilt somit

$$A_{s,s'}^j = \sum_{i=1}^{l-1} \text{Prob}(\pi_i^j = s, \pi_{i+1}^j = s'|x^j, \Theta). \tag{2.1}$$

Aus der erwarteten Anzahl $A_{s,s'}^j$ für eine Sequenz lässt sich hinterher durch Addition über alle Trainingsbeispiele und anschließender Normierung die Wahrscheinlichkeit $a_{s,s'}$ für diesen Zustandsübergang berechnen. Der Übersichtlichkeit halber wird im Folgenden auf die Mitführung der Modellparameter Θ verzichtet. Alle berechneten Wahrscheinlichkeiten beziehen sich jedoch stets auf die aktuellen Modellparameter.

Es bleibt die Frage, wie die $A_{s,s'}^j$ aus Gleichung 2.1 berechnet werden. Im Folgenden wird beschrieben, wie die einzelnen Summanden effektiv ermittelt werden können. Die Wahrscheinlichkeit, dass die Transition von s nach s' an der i-ten Position des während der Emission des Trainingsbeispiels x^j durchlaufenden Pfades benutzt wurde, beträgt

$$\begin{aligned}
\text{Prob}(\pi_i^j = s, \pi_{i+1}^j = s' | x^j) &= \frac{\text{Prob}(x^j, \pi_i^j = s, \pi_{i+1}^j = s')}{\text{Prob}(x^j)} \\
&= \frac{\text{Prob}(x_1^j \dots x_i^j, \pi_i^j = s) \cdot a_{s,s'} e_{s'}(x_{i+1}^j) \cdot \text{Prob}(x_{i+2}^j \dots x_l^j, \pi_{i+1}^j = s')}{\text{Prob}(x^j)} \\
&= \frac{f_s^j(i) \cdot a_{s,s'} e_{s'}(x_{i+1}^j) \cdot b_{s'}^j(i+1)}{\text{Prob}(x^j)}. \qquad (2.2)
\end{aligned}$$

Der erste Teil der Gleichungskette ergibt sich aus der Definition der bedingten Wahrscheinlichkeit. Im zweiten Schritt wird die Wahrscheinlichkeit, sich während der Ausgabe der Sequenz x^j an i-ter Stelle in Zustand s und an $(i+1)$-ter Stelle in Zustand s' zu befinden, weiter aufgespaltet: Zunächst müssen dazu die ersten i Stellen $x_1^j x_2^j \dots x_i^j$ der Ausgabesequenz emittiert und danach der Zustand s erreicht werden. Dies entspricht $\text{Prob}(x_1^j \dots x_i^j, \pi_i^j = s)$ und wird später mit $f_s^j(i)$ abgekürzt. Danach muss ein Übergang von s nach s' erfolgen und dort das $(i+1)$-te Symbol x_{i+1}^j der Sequenz ausgegeben werden. Die Wahrscheinlichkeit hierfür beträgt $a_{s,s'} \cdot e_{s'}(x_{i+1}^j)$. Schließlich muss noch ausgehend vom Zustand s' die restliche Sequenz $x_{i+2}^j, \dots, x_l^j$ emittiert werden. Dies geschieht mit der Wahrscheinlichkeit $\text{Prob}(x_{i+2}^j \dots x_l^j, \pi_{i+1}^j = s')$, die kurz mit $b_{s'}^j(i+1)$ bezeichnet wird. Dabei stehen $f_s^j(i)$ bzw. $b_s^j(i+1)$ für die sogenannten Forward- bzw. Backwardvariablen. Diese können durch die gleichnamigen Algorithmen 1 und 2, die im Folgenden vorgestellt werden, berechnet werden.

Algorithmus 1 Forward-Algorithmus

Eingabe: Symbolsequenz $x^j = x_1^j x_2^j \dots x_l^j$

1: $f_{s_0}^j(0) \leftarrow 1, \forall s \in \mathcal{S} \backslash \{s_0\} : f_s^j(0) \leftarrow 0, \forall i = 1, \dots, l : f_{s_0}^j(i) \leftarrow 0$ // Initialisierung
2: **for** $i = 1, \dots, l$ **do**
3: $\quad \forall s' \in \mathcal{S} \backslash \{s_0\} : f_{s'}^j(i) \leftarrow e_{s'}(x_i^j) \sum_{s \in \mathcal{S}} f_s^j(i-1) a_{s,s'}$
4: **end for**
5: $\text{Prob}(x^j) \leftarrow \sum_{s \in \mathcal{S}} f_s^j(l)$

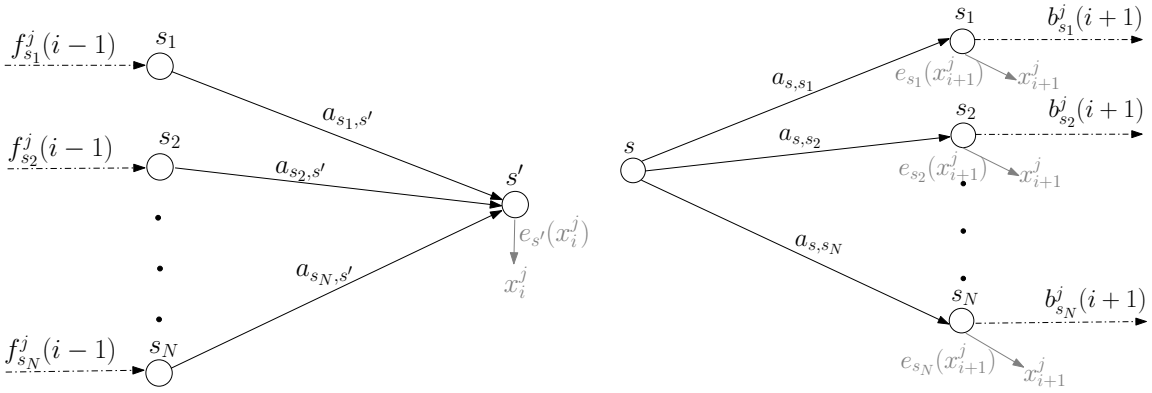
Die Initialisierung von $f_{s_0}^j(0)$ mit 1, während dieser Wert für alle $s \neq s_0$ auf 0 gesetzt wird, entspricht der Tatsache, dass ein Pfad nur mit dem Startzustand beginnen kann. Daraufhin wird die komplette Sequenz von vorne nach hinten durchlaufen und für jede Position i die Wahrscheinlichkeit $f_{s'}^j(i)$ bestimmt. Dies geschieht, indem für jeden möglichen Vorgängerzustand s die Wahrscheinlichkeit, zunächst in diesen Zustand zu gelangen und daraufhin einen Zustandsübergang von s nach s' durchzuführen, berechnet wird. Diese Werte werden für alle s addiert und noch mit der Wahrscheinlichkeit im Zustand s' das Symbol x_i^j auszugeben, multipliziert. Dies wird in Abbildung 2.2(a) noch einmal skizziert.

Algorithmus 2 Backward-Algorithmus

Eingabe: Symbolsequenz $x^j = x_1^j x_2^j \dots x_l^j$

1: $\forall s \in \mathcal{S}\backslash\{s_0\} : b_s^j(l) \leftarrow 1$ // Initialisierung
2: **for** $i = l-1, \dots, 1$ **do**
3: $\quad \forall s' \in \mathcal{S}\backslash\{s_0\} : b_s^j(i) \leftarrow \sum_{s' \in \mathcal{S}\backslash\{s_0\}} a_{s,s'} e_{s'}(x_{i+1}^j) b_{s'}^j(i+1)$
4: **end for**
5: $b_{s_0}^j(0) \leftarrow \sum_{s \in \mathcal{S}\backslash\{s_0\}} a_{s_0,s} e_s(x_1^j) b_s^j(1)$
6: $\text{Prob}(x^j) \leftarrow b_{s_0}^j(0)$

Beim Backward-Algorithmus erfolgt die Berechnung von hinten nach vorne. Initialisiert werden die $b_s^j(l)$-Werte für alle $s \neq s_0$ mit 1, da diese Zustände am Ende einer Sequenz erreicht werden können. Für den Startzustand gilt dies nicht, da er ausschließlich am Anfang einer Sequenz stehen darf. Danach werden die Werte $b_s^j(i)$ für die Position i der Sequenz und den Zustand s berechnet, indem die Wahrscheinlichkeiten von s in einen Nachfolgezustand s' überzugehen, dort das Symbol x_{i+1}^j auszugeben und dann die restliche Sequenz $x_{i+2}^j \dots x_l^j$, in Zustand s' beginnend, auszugeben, miteinander multipliziert und anschließend für alle s' addiert werden. Eine schematische Darstellung dieses Vorgehens findet sich in Abbildung 2.2(b).



(a) Forwardvariable $f_{s'}^j(i)$ (b) Backwardvariable $b_s^j(i)$

Abbildung 2.2: *Berechnung der Forward- und Backwardvariablen.* Die Forwardvariable $f_{s'}^j(i)$ setzt sich dabei aus den Werten für die Forwardvariablen $f_s^j(i-1)$ für $s \in \mathcal{S}$ zusammen. Analog setzt sich die Backwardvariable $b_s^j(i)$ aus den zuvor berechneten Backwardvariablen $b_{s'}^j(i+1)$ zusammen. Die Berechnung der Forwardvariablen erfolgt dabei von vorne nach hinten entlang der Sequenz, während die Sequenz zur Bestimmung der Backwardvariablen von hinten nach vorne abgearbeitet wird.

Die durch den Forward- und Backward-Algorithmus berechneten Werte für $\text{Prob}(x^j)$ stimmen natürlich überein, sie werden lediglich auf verschiedene Arten berechnet.

Ausgehend von Gleichung 2.2, berechnet sich die erwartete Anzahl $A_{s,s'}$ der Vorkommen von Übergängen von Zustand s in den Zustand s' in den Trainingsbeispielen $x^1, \ldots, x^k$ durch Addition über alle Trainingsbeispiele

$$\begin{aligned} A_{s,s'} &= \sum_{j=1}^{k} A^j_{s,s'} \\ &= \sum_{j=1}^{k} \frac{1}{\text{Prob}(x^j)} \sum_{i=1}^{l-1} f^j_s(i) a_{s,s'} e_{s'}(x^j_{i+1}) b^j_{s'}(i+1). \end{aligned} \tag{2.3}$$

Analog dazu ergibt sich die erwartete Anzahl $E_s(\gamma)$ der Vorkommen von Ausgabe des Symbols γ aus dem Zustand s:

$$E_s(\gamma) = \sum_{j=1}^{k} \frac{1}{\text{Prob}(x^j)} \sum_{(i|x^j_i=\gamma)} f^j_s(i) b^j_s(i). \tag{2.4}$$

Um von den Werten für $A_{s,s'}$ und $E_s(\gamma)$ wiederum auf die Modellparameter schließen zu können, müssen die Werte normiert werden. Für die Wahrscheinlichkeit $a_{s,s'}$ für eine Transition von einem Zustand s nach s' bzw. für die Emission eines bestimmten Symbols γ in s gilt also

$$a_{s,s'} = \frac{A_{s,s'}}{\sum\limits_{s'' \in \mathcal{S} \setminus \{s_0\}} A_{s,s''}} \text{ bzw. } e_s(\gamma) = \frac{E_s(\gamma)}{\sum\limits_{\gamma' \in \Gamma} E_s(\gamma')}. \tag{2.5}$$

Damit ist die Beschreibung einer Iteration des Baum-Welch-Algorithmus abgeschlossen. Jedoch ist noch ein geeignetes Gütemaß erforderlich, um ein Abbruchkriterium formulieren zu können. Hierzu wird die Wahrscheinlichkeit, mit der die Symbolsequenzen $x^1, x^2, \ldots, x^k$ von einem Modell mit den Parametern Θ ausgegeben werden, d. h.

$$L(\Theta) = \text{Prob}(x^1, x^2, \ldots x^k | \Theta) = \prod_{j=1}^{k} \text{Prob}(x^j | \Theta),$$

verwendet. Wie bereits erwähnt, ist das Ziel, die Likelihood-Funktion $L(\Theta)$ zu maximieren, also die Parameter Θ_{ML} mit

$$\begin{aligned} \Theta_{ML} &= \underset{\Theta}{\text{argmax}} \left\{ \prod_{j=1}^{k} \text{Prob}(x^j | \Theta) \right\} \\ &= \underset{\Theta}{\text{argmax}} \left\{ \text{Prob}(x^1, x^2, \ldots, x^k | \Theta) \right\} \end{aligned}$$

zu bestimmen. In der Statistik werden diese Parameter Θ_{ML} als Maximum Likelihood bezeichnet. Da gerade die Bestimmung der Parameter das eigentliche Problem darstellt, wird für den Rest dieser Arbeit, der Literatur folgend, die Funktion $L(\Theta)$ als Maximum Likelihood bezeichnet.

Die hierbei auftretenden Werte für die Wahrscheinlichkeiten sind im allgemeinen sehr klein, sodass bei der computerbasierten Berechnung der Werte oft Fehler durch den begrenzten Wertebereich auftreten. Durch Speicherung der logarithmierten Werte anstelle der Originalwerte wird dieser darstellbare Bereich auf Kosten der Darstellungsgenauigkeit vergrößert. Bei der logarithmierten Speicherung der einzelnen Werte wird dann ebenfalls die Log-Likelihood

$$LL(\Theta) = \log \text{Prob}(x^1, x^2, \ldots, x^k|\Theta) = \log \prod_{j=1}^{k} \text{Prob}(x^j|\Theta) = \sum_{j=1}^{k} \log \text{Prob}(x^j|\Theta) \quad (2.6)$$

verwendet. Da der Logarithmus eine streng monoton wachsende Funktion ist, kann auch der Logarithmus der Wahrscheinlichkeit maximiert werden, um die wahrscheinlichsten Parameter zu finden. Die Multiplikation der Werte wird dabei zu einer Addition.

Algorithmus 3 fasst die bisherigen Überlegungen zusammen.

Es kann gezeigt werden, dass in jeder Iteration des Baum-Welch-Algorithmus der Wert der Maximum Likelihood ansteigt und gegen ein lokales Maximum oder einen Stattelpunkt konvergiert (Baum 1972, Koski 2001).

Algorithmus 3 Baum-Welch-Algorithmus

Eingabe: Wähle beliebige Werte Θ für die Modellparameter

while Abbruchbedingung nicht erfüllt **do**
  $\forall s, s' \in \mathcal{S}, \gamma \in \Gamma : A_{s,s'} \leftarrow 0, E_{s,\gamma} \leftarrow 0$ // Initialisierung
  for $\forall$ Sequenzen $x^j = x_1^j x_2^j \ldots x_l^j$ **do**
    Berechne $f_k^j(i)$ für Sequenz j vermöge Forward-Algorithmus
    Berechne $b_k^j(i)$ für Sequenz j vermöge Backward-Algorithmus
    $\forall s \in \mathcal{S}, s' \in \mathcal{S}\backslash\{s_0\} : A_{s,s'} \leftarrow A_{s,s'} + \frac{1}{\text{Prob}(x^j)} \sum_{i=1}^{l-1} f_s^j(i) a_{s,s'} e_{s'}(x_{i+1}^j) b_{s'}^j(i+1)$
    $\forall s \in \mathcal{S}\backslash\{s_0\}, \gamma \in \Gamma : E_{s,\gamma} \leftarrow E_{s,\gamma} + \frac{1}{\text{Prob}(x^j)} \sum_{\{i|x_i^j=\gamma\}} f_s^j(i) b_s^j(i)$
  end for
  Berechne neue Modellparameter Θ' gemäß Gleichung 2.5
  Berechne die Log-Likelihood für Θ' gemäß Gleichung 2.6
end while

Als Abbruchbedingung kann hierbei beispielsweise eine vorgegebene Anzahl Schleifendurchläufe gewählt werden oder es wird abgebrochen, wenn die Änderung der Log-Likelihood einen festgesetzten Wert unterschreitet.

2.2.3 Viterbi-Algorithmus

Der in diesem Abschnitt beschriebene Viterbi-Algorithmus stellt die verbreitetste Möglichkeit dar, aus einer gegebenen Sequenz von beobachteten Symbolen $x = x_1 x_2 \dots x_l$ den Pfad π^* zu berechnen, der dieser Sequenz am wahrscheinlichsten zugrunde liegt. Dieser Vorgang wird auch Dekodierung genannt. Dieser Pfad soll später dazu genutzt werden, die Markierungen zu den einzelnen Nukleotiden zu bestimmen und so die CpG-Inseln vorherzusagen.

Der Viterbi-Algorithmus verwendet — wie auch die bisher vorgestellten Algorithmen — den Ansatz der dynamischen Programmierung. Dabei wird die Tatsache ausgenutzt, dass sich ein Pfad $\Pi = \pi_0, \dots, \pi_{l-1}, \pi_l$ aus einem Teilpfad $\Pi' = \pi_0, \dots, \pi_{l-1}$ und einem darauf folgenden Übergang von π_{l-1} nach π_l zusammensetzt. Sei $v_s(i)$ die Wahrscheinlichkeit des wahrscheinlichsten Pfades der Länge i, der im Zustand s mit dem beobachteten Symbol x_i endet.

Somit ergibt sich die Wahrscheinlichkeit $v_s(i)$, indem für jeden möglichen Vorgängerzustand s' bestimmt wird, wie wahrscheinlich es ist, dass dieser zuvor erreicht wurde und daraufhin ein Übergang von s' in s erfolgte. Das Maximum dieser Werte wird gewählt und es wird berücksichtigt, dass daraufhin im Zustand s noch das Symbol x_i ausgegeben werden muss. Es ergibt sich also die Bellman'sche Optimalitätsgleichung

$$v_s(i) = e_s(x_i) \cdot \max_{s'} \{v_{s'}(i-1) a_{s',s}\} .$$

Da nur im Startzustand gestartet werden kann, gilt $v_{s_0}(0) = 1$ und $v_s(0) = 0$ für alle anderen Zustände $s \neq s_0$. Auf diese Art können sukzessive die Werte $v_s(1), v_s(2), \dots, v_s(l)$ berechnet werden. Algorithmus 4 zeigt den vollständigen Viterbi-Algorithmus.

Algorithmus 4 Viterbi-Algorithmus

Eingabe: Symbolsequenz $x = x_1 x_2 \dots x_l$

1: $v_{s_0}(0) \leftarrow 1, \forall s \in \mathcal{S} \backslash \{s_0\} : v_s(0) \leftarrow 0, \forall i = 1, \dots, l : v_{s_0}(i) \leftarrow 0$ // Initialisierung
2: **for** $i = 1, \dots, l$ **do**
3: $\quad \forall \in \mathcal{S} \backslash \{s_0\} : v_s(i) \leftarrow e_s(x_i) \cdot \max_{s'} (v_{s'}(i-1) a_{s',s})$
4: $\quad \forall \in \mathcal{S} \backslash \{s_0\} : ptr_i(s) \leftarrow \text{argmax}_{s'}(v_{s'}(i-1) a_{s',s})$
5: **end for**
6: $\pi_l^* \leftarrow \text{argmax}_s(v_s(l))$
7: **for** $i = l, \dots, 1$ **do**
8: $\quad \pi_{i-1}^* \leftarrow ptr_i(\pi_i^*)$
9: **end for**

Um nicht nur die Wahrscheinlichkeiten zu berechnen, sondern letztendlich auch in der Lage zu sein, den Pfad $\Pi^* = \pi_0^*, \pi_1^*, \dots, \pi_l^*$ zu rekonstruieren, wird zu jedem $v_s(i)$ zusätzlich ein Zeiger $ptr_i(s)$ gespeichert, der markiert, von welchem Zustand aus an dieser Stelle der Sequenz der Zustand s erreicht wurde.

2.2.4 HMMs für Sequenzen mit Markierungen

Die bisher betrachteten HMMs sind zwar in der Lage, Nukleotidsequenzen zu erzeugen, jedoch noch nicht, Markierungen für diese Nukleotide zu bestimmen. Um die in Abschnitt 2.1 vorgestellte Problemstellung bearbeiten zu können, müssen die HMMs dementsprechend erweitert werden. Zu jedem einzelnen Ausgabesymbol x_i^j, in diesem Fall also eines der vier Nukleotide A, C, G oder T, muss eine Markierung y_i^j berücksichtigt werden, die festlegt, ob dieses Nukleotid innerhalb oder außerhalb einer CpG-Insel liegt. Es müssen also die oben beschriebenen Algorithmen so angepasst werden, dass sie mit derartigen — in Abbildung 2.3 dargestellten — Trainingsbeispielen umgehen können.

DNA-Sequenz `GGGTCCGCGGGGAGGGGGCGCGCTTGGGAACCGGGAGGCCCCGCGAGATGCCGCC`
CpG-Inseln `00000000000000000000000000000000000000000000000000011111`

DNA-Sequenz `GGGTCCGCGGGGAGGGGGCGCGCTTGGGAACCGGGAGGCCCCGCGAGATGATTAC`
CpG-Inseln `11111111111111111111111111111111111111111111111100000`

Abbildung 2.3: *Beispiel für eine DNA-Sequenz mit Markierungen für CpG-Inseln.* Die DNA-Sequenz beschreibt dabei die Ausgabesequenz. Die CpG-Inseln werden durch die Markierungssequenz festgelegt. Eine Markierung von 1 zeigt an, dass das zugehörige Nukleotid in der Ausgabesequenz innerhalb einer CpG-Insel liegt, eine 0 markiert Nukleotide außerhalb von CpG-Inseln.

Für jeden Zustand s des HMMs wird nun eine Markierung label(s) festgelegt. In diesem Fall steht eine 1 für Zustände, die zu CpG-Inseln gehören und eine 0 für die restlichen Zustände. Später können an diesen Markierungen der Zustände die CpG-Inseln abgelesen werden: Befindet sich das HMM in einem Zustand mit der Markierung 1, liegt das zugehörige Ausgabesymbol innerhalb einer CpG-Insel; ansonsten nicht. Um bei den Algorithmen nicht durch eine Fallunterscheidung gewährleisten zu müssen, dass der Startzustand nicht von einem anderen Zustand aus erreicht werden kann, wird dieser bei der Implementierung eine eigene, sich von den übrigen verwendeten unterscheidende Markierung erhalten. So ist keine Sonderbehandlung des Startzustandes notwendig. Dies ermöglicht eine elegantere Handhabung des Startzustandes.

Die Algorithmen 1 bis 3 werden nun dahingehend modifiziert, dass sie nur Pfade zulassen, auf denen die Markierung y_i^j des Trainingsbeispiels der Markierung label(s) des Zustandes s entspricht, in dem sich das HMM an dieser Stelle befindet. Diese Pfade werden auch als gültige Pfade bezeichnet. Ungültige Pfade erhalten die Wahrscheinlichkeit 0. Für die Algorithmen bedeutet das, dass alle Forward- und Backwardvariablen, die nicht auf einem gültigen Pfad liegen, auf 0 gesetzt werden. Die restlichen Werte werden wie oben beschrieben berechnet. Die Berechnung dieser Variablen wird durch den Forward- bzw. Backward-Algorithmus übernommen, die nötigen Modifizierungen werden in Algorithmus 5 und 6 beschrieben.

An dieser Stelle sei erwähnt, dass in der bisherigen Version des Baum-Welch-Algorithmus *alle* Pfade als gültig betrachtet werden. Somit reicht zur Anpassung des Baum-Welch-Algorithmus aus, die modifizierten Algorithmen 5 und 6 anstelle der Algorithmen 1 und 2 zur Berechnung der Forward- und Backwardvariablen zu verwenden. Zur Vorhersage der Markierungen zu einer Sequenz wird weiterhin der Viterbi-Algorithmus benutzt. Die ausgegebene Markierungssequenz entspricht den Markierungen der Zustandsfolge auf dem wahrscheinlichsten Pfad Π^*.

Algorithmus 5 Modifizierter Forward-Algorithmus für Sequenzen mit Markierungen

Eingabe: Symbolsequenz $x^j = x_1^j x_2^j \dots x_l^j$ mit Markierungen $y^j = y_1^j y_2^j \dots y_l^j$

1: $f_{s_0}^j(0) \leftarrow 1, \forall s \in \mathcal{S}\backslash\{s_0\} : f_s(0) \leftarrow 0, \forall i = 1, \dots, l : f_{s_0}^j(i) \leftarrow 0$ // Initialisierung

2: **for** $i = 1, \dots, l$ **do**

3: $$\forall s' \in \mathcal{S} : f_{s'}^j(i) \leftarrow \begin{cases} e_{s'}(x_i^j) \sum\limits_{s \in \mathcal{S}} f_s^j(i-1) a_{s,s'} & \text{falls label}(s') = y_i^j \\ 0 & \text{falls label}(s') \neq y_i^j \end{cases}$$

4: **end for**

5: $\text{Prob}(x^j, y^j) \leftarrow \sum\limits_s f_s^j(l)$

Algorithmus 6 Modifizierter Backward-Algorithmus für Sequenzen mit Markierungen

Eingabe: Symbolsequenz $x^j = x_1^j x_2^j \dots x_l^j$ mit Markierungen $y^j = y_1^j y_2^j \dots y_l^j$

1: $$\forall s \in \mathcal{S} : b_s^j(l) \leftarrow \begin{cases} 1 & \text{falls label}(s) = y_l^j \\ 0 & \text{falls label}(s) \neq y_l^j \end{cases}$$ // Initialisierung

2: **for** $i = l-1, \dots, 1$ **do**

3: $$\forall s \in \mathcal{S} : b_s^j(l) \leftarrow \begin{cases} \sum\limits_{s' \in \mathcal{S}\backslash\{s_0\}} a_{s,s'} e_{s'}(x_{i+1}^j) b_{s'}^j(i+1) & \text{falls label}(s) = y_l^j \\ 0 & \text{falls label}(s) \neq y_l^j \end{cases}$$

4: **end for**

5: $b_{s_0}^j(0) \leftarrow \sum\limits_s a_{s_0,s} e_s(x_1^j) b_s^j(1)$

6: $\text{Prob}(x^j, y^j) \leftarrow b_{s_0}^j(0)$

Für die Vorhersage von CpG-Inseln sind die Markierungen der Zustände von weitaus größerer Bedeutung als die durchlaufenen Zustände selbst. Daher sollte anstelle der Wahrscheinlichkeit $\text{Prob}(x^j|\Theta)$, dass die Sequenz x^j vom Modell erzeugt wird, die Wahrscheinlichkeit $\text{Prob}(y^j|x^j, \Theta)$, dass die Markierungen y^j bei vorgegebener Sequenz x^j vorliegen, maximiert werden. Dies wird als die Conditional Maximum Likelihood bezeichnet (kurz: CML, siehe dazu auch Rabiner 1989). Praktisch lässt sich die CML mithilfe der folgenden Gleichung berechnen.

$$\text{Prob}(y^j|x^j, \Theta) = \frac{\text{Prob}(y^j, x^j|\Theta)}{\text{Prob}(x^j|\Theta)} \tag{2.7}$$

Diese Identität folgt unmittelbar aus der Definition der bedingten Wahrscheinlichkeit. Die Berechnung von $\text{Prob}(y^j, x^j|\Theta)$ erfolgt dabei beispielsweise mit dem Backward-Algorithmus für Sequenzen mit Markierungen (Algorithmus 6) und die von $\text{Prob}(x^j|\Theta)$ mit dem gewöhnlichen Backward-Algorithmus (Algorithmus 2). Da für die Berechnung alle Werte logarithmiert verwaltet werden, wird aus dem Quotienten in Gleichung 2.7 eine Differenz. Wünschenswert wäre ein monotones Ansteigen der CML während der Ausführung des Baum-Welch-Algorithmus. Allerdings ist dies nicht der Fall. Ebenso ist kein Algorithmus bekannt, der eine solche Monotonie garantiert.

Weitere Details über HMMs für Sequenzen mit Markierungen finden sich beispielsweise in Durbin u. a. (1998) und Krogh (1994).

2.2.5 Evolutionäre Algorithmen

In diesem Abschnitt wird das Prinzip der evolutionären Algorithmen eingeführt. Dies sind randomisierte Suchheuristiken, die auf einer geeigneten Bewertung der aktuell gefundenen Lösung basieren. Der Optimierungsprozess kann durch die Bewertungsfunktion gesteuert werden. Auf diese Art und Weise bieten evolutionäre Algorithmen die Möglichkeit, zusätzlich zu den Zustandsübergangs- und Ausgabewahrscheinlichkeiten ebenfalls die Topologie zu modifizieren. Ebenso könnte beispielsweise auch die CML in die Bewertung eingehen.

In der Biologie ist der Begriff der Evolution weithin bekannt. Er wurde von Charles Darwin (*1809 – †1882) geprägt, der in seiner Evolutionstheorie die Auffassung vertrat, dass alle Lebewesen sich im Laufe der Zeit kontinuierlich verändern und so langsam an die vorherrschenden Umgebungsbedingungen anpassen. In diesem Zusammenhang ist auch das „Survival of the fittest“ — das Überleben des Bestangepassten — zu sehen. Lebewesen, die sich gut an ihre Umgebung angepasst haben, vermehren sich häufiger und geben diese vorteilhaften Eigenschaften an ihre Nachkommen weiter. Schlecht angepasste Lebewesen vermehren sich weniger oft und werden so nach und nach verdrängt.

Evolutionäre Algorithmen greifen dieses Prinzip auf. Sie verwalten zu jedem Zeitpunkt eine Menge von Suchpunkten des Lösungsraums. Traditionell werden die Suchpunkte als Individuen und die verwaltete Menge als Population bezeichnet. Die Individuen werden basierend auf ihren Eigenschaften durch die sogenannte Fitnessfunktion bewertet. Ein Selektionsoperator wählt eine Subpopulation aus, aus der Nachfolgeindividuen generiert werden. Nach einer erneuten Bewertung mittels der Fitnessfunktion wird aus der Gesamtheit die neue Population selektiert. Eine solche Iteration wird Generation genannt. Dieser Ansatz wurde bereits Anfang der sechziger Jahre von Schwefel und Rechenberg vorgestellt und erfolgreich auf Probleme aus den verschiedensten Bereichen angewendet (siehe z. B. Rechenberg 1973 und Schwefel 1975). Seit ihrer Einführung hat sich ein breites Spektrum an Variationen entwickelt.

Um einen evolutionären Algorithmus formal zu beschreiben, werden zunächst die folgenden Begriffe geklärt:

- Repräsentation der Individuen
- Mutations- und Rekombinationsoperatoren
- Selektionsoperator
- Fitnessfunktion

Die Repräsentation muss für die vorliegende Problemstellung in der Lage sein, ein HMM eindeutig zu beschreiben. Dazu sind insbesondere die Zustandsübergangs- und Ausgabewahrscheinlichkeiten nötig. Diese können in Form von zwei Matrizen gespeichert werden. Aus deren Dimensionen ergeben sich dann auch indirekt die Anzahlen der Zustände und der Ausgabesymbole. Zusätzlich ist es bei HMMs für Sequenzen mit Markierungen notwendig, die Markierungen der einzelnen Zustände, beispielsweise in Form eines Vektors, zu verwalten. Es ergibt sich die folgende Definition eines Individuums.

Definition 2. *Ein Individuum $\mathcal{I}$ besteht aus einem Tripel (ℓ, A, E) mit*

- *Vektor $\ell = (\ell_0, \ell_1, \ldots, \ell_n)$ der Zustandsmarkierungen; ℓ_i bezeichnet dabei die Markierung des i-ten Zustands, die Markierung ℓ_0 des Startzustands s_0 ist fest und unterscheidet sich von den übrigen verwendeten Markierungen,*
- *$(n+1) \times n$ Matrix A der Übergangswahrscheinlichkeiten; $a_{s,s'}$ bezeichnet dabei die Wahrscheinlichkeit vom Zustand s in den Zustand s' zu gelangen,*
- *$n \times m$ Matrix E der Ausgabewahrscheinlichkeiten; $e(s, \gamma)$ beziffert die Wahrscheinlichkeit im Zustand s ein bestimmtes Symbol $\gamma \in \Gamma$ zu emittieren.*

Eine Population bezeichnet damit eine Menge P_t von Individuen zu einem bestimmten Zeitpunkt t. Unter all den Möglichkeiten die Subpopulation für die Mutation und Rekombination zu wählen, werden in dieser Arbeit aufgrund der geringen Populationsgrößen alle Individuen der Population zur Erzeugung der Nachfolgeindividuen verwendet. Aus den einzelnen Individuen der aktuellen Population werden nun in jeder Generation durch Mutation und Rekombination neue Individuen erzeugt. Welche dieser neu erzeugten Individuen und der Individuen aus der aktuellen Population P_t die Population zum Zeitpunkt $t+1$ ausmachen, wird durch den Selektionsoperator bestimmt. Dies ist in Algorithmus 7 zusammengefasst.

Wie genau Mutations-, Rekombinations- und Selektionsoperator arbeiten, kann — und muss — problemspezifisch festgelegt werden. Lediglich die grundlegende Funktionalität bleibt gleich. So erzeugt ein Mutationsoperator aus einem Individuum ein neues, während ein Rekombinationsoperator aus zwei oder mehr Individuen wiederum eine

bestimmte Anzahl an neuen generiert. Zur Erzeugung eines Individuums werden aus den potentiellen Eltern, was hier der gesamten aktuellen Population entspricht, gleichverteilt zufällig die Eltern bestimmt, die für die Mutation und die Rekombination verwendet werden.

Algorithmus 7 Evolutionärer Algorithmus

1: Initialisiere Population P_0
2: Evaluiere P_0
3: $t \leftarrow 1$
4: **while** Abbruchbedingung nicht erfüllt **do**
5: $P' \leftarrow mut(P_{t-1})$
6: $P'' \leftarrow rek(P')$
7: Evaluiere P''
8: $P_t \leftarrow sel(P'' \cup P_{t-1})$
9: $t \leftarrow t + 1$
10: **end while**

Die Evaluierung der Populationen wird durch die Fitnessfunktion übernommen. Diese berechnet zu jedem Individuum der aktuellen Population einen Fitnesswert, der dann als Maß für die Güte des entsprechenden Individuums verwendet wird. Diese muss natürlich in jedem Fall auf die Problemstellung angepasst werden, da sie letztendlich entscheidenden Einfluss darauf hat, welche Individuen für die Population in der nächsten Generation ausgewählt werden. Im Verlauf dieser Arbeit werden verschiedene Fitnessfunktionen entwickelt und untersucht.

Da hier — wie bereits erwähnt — die gesamte Population für die Erzeugung neuer Individuen herangezogen wird, ist lediglich noch die Festlegung des Selektionsoperators für die Auswahl der Nachfolgepopulation nötig. Auch hierfür gibt es verschiedenste Möglichkeiten der Realisierung. Es werden dabei zwei Klassen von Selektionsoperatoren unterschieden. Bei der Plus-Selektion stehen die alte Population sowie die neu erzeugten Individuen zur Auswahl. Die Komma-Selektion hingegen verwirft die alte Population vollständig und wählt lediglich zwischen den neu erzeugten Individuen. Hier werden ausschließlich Plus-Selektionen verwendet. Dies entspricht der Zeile 8 in Algorithmus 7. Auch für den Ablauf von Plus-Selektionen gibt es verschiedene Möglichkeiten, die von einer deterministischen Auswahl der am besten bewerteten Individuen bis hin zu stochastischen Varianten reichen, die dem Einfluss von zufälligen Ereignissen auf das Überleben von Lebewesen näher kommen. Als verbreitete Variante sei hier die Rouletteradselektion erwähnt. Diese teilt jedem Individuum proportional zu seiner Fitness einen Anteil auf einem Rouletterad zu. Danach wird durch wiederholtes Drehen am Rad die gewünschte Anzahl Individuen für die Nachfolgegeneration ermittelt. Formal ist dies in Algorithmus 8 dargestellt.

Algorithmus 8 Rouletteradselektion

Eingabe: Fitnesswerte $\Psi(\mathcal{I}_1), \Psi(\mathcal{I}_2), \ldots, \Psi(\mathcal{I}_s)$ der Individuen, Anzahl k der zu selektierenden Individuen

1: Berechne $\forall i = 1, \ldots, s : \varphi(i)$ gemäß Gleichung 2.8
2: **for** $i = 1, \ldots, k$ **do**
3: $j = 1$
4: $sum \leftarrow \varphi(j)$
5: Wähle $r \in [0, 1)$ uniform zufällig
6: **while** $sum < r$ **do**
7: $j \leftarrow j + 1$
8: $sum \leftarrow sum + \varphi(j)$
9: Füge $\mathcal{I}_j$ zu selektierten Individuen hinzu
10: **end while**
11: **end for**

Die Auswahlwahrscheinlichkeit $\varphi(i)$ für das Individuum $\mathcal{I}_i$ ergibt sich proportional zu seinem Fitnesswert $\Psi(\mathcal{I}_i)$ durch

$$\varphi(i) = \varphi(\mathcal{I}_i) = \frac{\Psi(\mathcal{I}_i)}{\sum_j \Psi(\mathcal{I}_j)}. \tag{2.8}$$

Bei diesem Verfahren ist die Wahrscheinlichkeit, dass ein Individuum mit hoher Fitness die Selektion „überlebt“, größer als bei einem Individuum mit geringer Fitness. Dennoch ist nicht gewährleistet, dass das Individuum mit dem besten Fitnesswert tatsächlich selektiert wird. Ebenso ist es möglich, dass Individuen mehrfach ausgewählt werden und demzufolge mehrfach in der nächsten Generation vertreten sind.

Eine leichte Abwandlung der Rouletteradselektion stellt das stochastische universelle Sampling dar. Dieses benötigt zur Bestimmung einer vollständigen Nachfolgegeneration mit k Individuen lediglich *eine* Zufallszahl. Hier wird wiederum jedem Individuum ein zu seiner Fitness proportionaler Teil des Rouletterades zugeordnet. Danach wird gleichverteilt eine Zufallszahl $r \in [0, \frac{1}{k})$ erzeugt. Diese legt die Stelle auf dem Rouletterad fest, an der sich das erste zu wählende Individuum befindet, nämlich $r \cdot 360°$. Die weiteren Stellen und damit die Individuen, die gewählt werden sollen, werden festgelegt, indem ausgehend von r immer eine $\frac{1}{k}$-tel Umdrehung weitergegangen wird, also $\frac{1}{k} \cdot 360°$. Dies ist in Abbildung 2.4 beispielhaft skizziert. Dieses Verfahren stellt eine Laufzeitverbesserung dar, da nicht für jedes auszuwählende Individuum ein Durchlauf durch die Elternindividuen nötig ist, sondern lediglich ein Durchlauf zur Festlegung aller Nachfolgeindividuen. Allerdings ist es auch beim stochastischen universellen Sampling möglich, dass ein Individuum mehrfach für die nächste Generation ausgewählt wird.

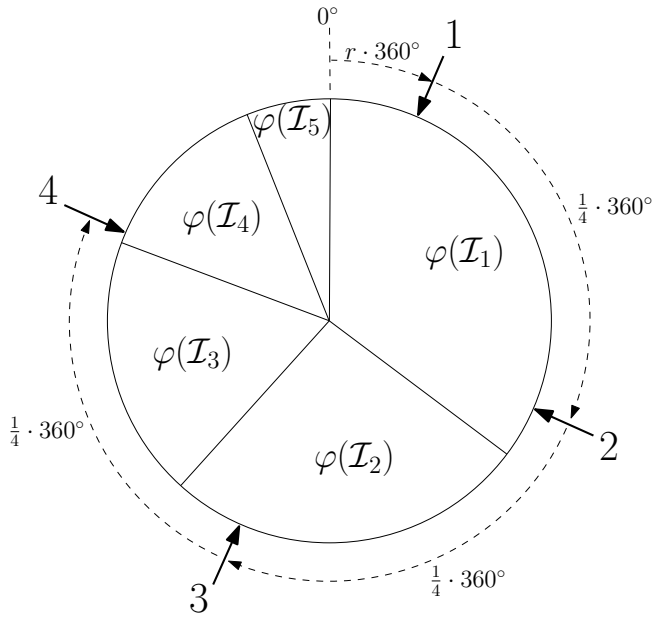


Abbildung 2.4: *Beispiel für das stochastische universelle Sampling mit* $k = 4$. Die Nachfolgepopulation besteht aus den Individuen $\mathcal{I}_1$, $\mathcal{I}_1$, $\mathcal{I}_2$ und $\mathcal{I}_4$. Das Individuum $\mathcal{I}_1$ wird dabei zweimal selektiert und ist demzufolge in der Nachfolgegeneration zweimal enthalten.

Kapitel 3

Betrachtungen der klassischen Ansätze

Im Folgenden wird zunächst eine Implementierung des Baum-Welch-Algorithmus geschaffen. Anhand dieser lässt sich der Verlauf der Maximum Likelihood (ML) und der Conditional Maximum Likelihood (CML) über die Iterationen des Baum-Welch-Algorithmus betrachten und kann somit zu Vergleichszwecken herangezogen werden. Schließlich werden noch eine Möglichkeit, das resultierende HMM zu bewerten sowie eine Alternative zum Viterbi-Algorithmus für die Ermittlung der Markierungen entlang einer Eingabesequenz vorgestellt.

3.1 Implementierung

Die Implementierung des Baum-Welch-Algorithmus sowie der noch folgenden Algorithmen wird in der Programmiersprache MATLAB vorgenommen. Obwohl bereits eine Vielzahl von MATLAB Implementierungen für HMMs frei verfügbar sind, ist keine davon in der Lage, mit HMMs mit Markierungen umzugehen, sodass eine von Grund auf neue Implementierung geschaffen wird.

3.1.1 Logarithmische Speicherung der Wahrscheinlichkeiten

Bei der computergestützten Berechnung von Gleitkommazahlen wird eine Zahl $\pm m \cdot 2^e$ durch Vorzeichen, Mantisse m und Exponent e gespeichert. Dabei steht für die einzelnen Elemente eine gewisse Anzahl von Bits zur Verfügung. In der IEEE-754 Norm werden 52 Bits für die Mantisse, 11 Bits für den Exponenten und 1 Bit für das Vorzeichen für einen `double`-Wert vorgesehen. Dieser Konvention folgt auch MATLAB.

Wie bereits in Kapitel 2.2.2 erwähnt, wird in der Literatur die Speicherung von logarithmierten Werten empfohlen, um den Wertebereich der Wahrscheinlichkeiten zu vergrößern, denn nur so ist die Berechnung der — zumeist sehr kleinen — Wahrscheinlichkeitswerte möglich. Für die in den Algorithmen vorkommenden Multiplikationen stellt

dies kein Problem dar. Aufgrund des Logarithmengesetzes $\log(p \cdot q) = \log(p) + \log(q)$ ist sofort klar, dass die Multiplikation der ursprünglichen Werte zu einer Addition der logarithmierten Werte wird. Allerdings enthalten der Baum-Welch-Algorithmus (Algorithmus 3) sowie die Algorithmen 5 und 6 auch jeweils eine Summe und es gibt kein ähnliches Gesetz für $\log(p+q)$. Demzufolge müssen die logarithmierten Werte zunächst in die Werte, die sie repräsentieren, konvertiert werden, um deren Summe berechnen zu können. Im Folgenden wird geklärt, wie dies geschehen kann, ohne erneut auf das Wertebereichsproblem zu stoßen.

Alle Werte $a_{s,s'}$ und $e_s(\gamma)$ werden logarithmiert[1] gespeichert. Die Logarithmen der Forward- und Backwardvariablen werden dann ebenfalls direkt berechnet. Die Berechnung der Summen erfolgt wie im Folgenden am Beispiel der Zeile 3 des modifizierten Forward-Algorithmus beschrieben. Für die anderen Algorithmen ist selbstverständlich eine analoge Vorgehensweise möglich.

Durch die kanonische Konvertierung der logarithmierten Werte entstünden in vielen Fällen Werte, die zu klein sind, um noch mit der oben beschriebenen Darstellung ausgedrückt werden zu können. Die grundsätzliche Idee ist nun, die Werte zunächst in einen Bereich zu verschieben, in dem auch die ursprünglichen Werte noch darstellbar sind, dort die gewünschten Berechnungen durchzuführen — in diesem Fall also die Summenbildung — und sie daraufhin wieder in den alten Wertebereich zurück zu verschieben. Dies wird durch die im Folgenden beschriebenen Schritte realisiert.

Zunächst werden die einzelnen logarithmierten Summanden berechnet. In diesem speziellen Fall werden dazu also $\tilde{d}_s \leftarrow f_s^j(i-1) + a_{s,s'}$ für alle $s \in \mathcal{S}$ bestimmt, wobei es sich dabei um die logarithmisch abgespeicherten Variablenwerte handelt. Liegen die Summanden vor, wird ihr Maximum D ermittelt.

Der erste Schritt der Addition gemäß den Vorgaben der IEEE-Norm besteht darin, die Exponenten der beiden Zahlen durch Verschieben der Mantisse aneinander anzupassen. Daher beeinflussen sich nur Werte, deren Exponenten sich nicht um mehr als die Anzahl $\|m\|$ der für die Speicherung der Mantisse vorgesehenen Bits unterscheiden. Es müssen nun also nur solche Summanden berücksichtigt werden, deren repräsentierte Werte sich um nicht mehr als einen Faktor von 2^{52} vom Maximum unterscheiden. Alle anderen Werte würden ohnehin keinen Effekt auf das Ergebnis haben. Auf diese Art müssen meist nur wenige Summanden berücksichtigt werden, ohne aber zusätzliche Einbußen in der Genauigkeit in Kauf nehmen zu müssen. An dieser Stelle ist es sehr hilfreich, den Logarithmus zur Basis 2 zu verwenden, da in diesem Fall die Überprüfung durch einen Vergleich der $\tilde{d}_s$ mit $D - 52$ realisiert werden kann.

Im nächsten Schritt werden die verbleibenden Summanden um D verschoben. Nun liegt ihr Maximum bei 0 und die restlichen Werte im Intervall $[-52, 0]$. Auf diese Weise ist

[1] Ist im Folgenden von Logarithmen die Rede, so ist stets der Logarithmus zur Basis 2 gemeint.

sichergestellt, dass bei der Berechnung der ursprünglichen Werte keine Underflows, also Werte, die kleiner als die kleinste darstellbare Zahl sind, auftreten können. Durch Bildung von $2^{\tilde{d}_s}$ können nun die einzelnen benötigten Werte ermittelt und dann summiert werden. Von dem Ergebnis wird dann wieder der Logarithmus gebildet und er wird durch die Addition des Maximums D wieder zurück verschoben.

Die Korrektheit der Rechnung folgt unmittelbar aus folgender Gleichungskette:

$$\begin{aligned} D + \log_2 \sum_{s=1}^{n} 2^{\tilde{d}_s - D} &= D + \log_2 \sum_{s=1}^{n} 2^{\tilde{d}_s} \cdot 2^{-D} \\ &= D + \log_2 \left(2^{-D} \cdot \sum_{s=1}^{n} 2^{\tilde{d}_s} \right) \\ &= D - D + \log_2 \sum_{s=1}^{n} 2^{\tilde{d}_s} \\ &= \log_2 \sum_{s=1}^{n} 2^{\tilde{d}_s} \end{aligned} \tag{3.1}$$

Damit ist die Summe berechnet. Um nun die Berechnung der Zeile 3 des modifizierten Forward-Algorithmus abzuschließen, wird der Wert $e_{s'}(x_i)$ dazu addiert. In den anderen Fällen, wo Summen in den Algorithmen auftreten, ist eine analoge Vorgehensweise natürlich immer möglich und wird in der Implementierung genutzt.

Algorithmus 9 Summation über nicht logarithmierte Werte

Eingabe: Logarithmierte Summanden $d_1, d_2, \ldots, d_n$

1: $D \leftarrow \max\{d_s\}$
2: Entferne alle d_s mit $d_s < D - 52$
3: Für alle verbleibenden Elemente $\tilde{d}_s \leftarrow d_s - D$
4: $erg \leftarrow \sum_{s=1}^{n} 2^{\tilde{d}_s}$
5: $erg \leftarrow erg + D$

3.2 Trainingsdaten

Die in Kapitel 2.1.1 erwähnten Beispieldaten werden in zwei Mengen geteilt, um einen Teil für das Training und den anderen hinterher zur Bewertung des Verhaltens des HMMs zu benutzen. Dies soll die Unabhängigkeit zwischen den Trainings- und den Validierungsdaten sicherstellen. Dadurch ist es möglich, eine Aussage über die Güte der vorhergesagten CpG-Inseln auf unbekannten Daten zu treffen. Um nun nicht durch die Auswahl jedes i-ten Trainingsbeispiels eventuelle Regelmäßigkeiten aus den Daten zu

übernehmen, wird die Aufteilung zufällig durchgeführt. Hier wird $\frac{1}{10}$ der Beispieldaten in die Validierungsdatenmenge für die spätere Bewertung übernommen und nur die verbleibenden $\frac{9}{10}$ für das Training des HMMs verwendet. Dazu werden die Indices der Beispiele für die Bewertung gleichverteilt zufällig bestimmt und dann die dazugehörigen Beispielsequenzen aus der Menge der Trainingsbeispiele entfernt und separat gespeichert.

3.3 Entwicklung der ML und der CML

Als Ausgangspunkt für die Anpassung der Modellparameter Θ durch den Baum-Welch-Algorithmus dient ein HMM, das zufällig initialisiert wird. Für dieses HMM werden 80 Iterationen des Baum-Welch-Algorithmus berechnet. Jede Iteration erzeugt dabei ein HMM mit neuen Modellparametern. Im Anschluss an jede Iteration werden dann sowohl die logarithmierte Maximum Likelihood als auch die Conditional Maximum Likelihood gemäß Gleichung 2.6 bzw. 2.7 bestimmt. Die Ergebnisse dieses Laufs sind in den beiden nachfolgenden Abbildungen 3.1 und 3.2 dargestellt.

Abbildung 3.1 zeigt den zeitlichen Verlauf der logarithmierten Maximum Likelihood. Erwartungsgemäß steigt der Wert der logarithmierten Maximum Likelihood bei jeder der 80 Iterationen kontinuierlich an. Da die logarithmierten Wahrscheinlichkeiten über alle Trainingsbeispiele addiert werden und offensichtlich alle negativ sind, sind die Werte der logarithmierten ML allesamt relativ klein. Die Werte sind augenscheinlich direkt von der Anzahl der Trainingsbeispiele abhängig: Je mehr Beispiele vorliegen, desto kleiner ist das Gesamtergebnis. Wird nun der Durchschnitt über alle 1396 Trainingssequenzen gebildet, ergeben sich logarithmierte ML-Werte, die während der Berechnung von -7280 auf -7123 angestiegen sind. Dies entspricht einer Steigerung der Wahrscheinlichkeit von 2^{-7280} auf 2^{-7123}.

Folgende Betrachtungen sollen bei der Einordnung der Größenordnung helfen: Die vorgegebenen Beispielsequenzen sind im Durchschnitt 3662 Zeichen lang. Einzelne Beispielsequenzen erreichen sogar eine Länge von 185775 Nukleotiden. Es gibt 4^{3662} Ausgabesymbolsequenzen der Länge 3662. Im Falle von gleichverteilten Sequenzen hat jede einzelne eine Auftrittswahrscheinlichkeit von 2^{-7320}.

Die während der nach 80 Iterationen des Baum-Welch-Algorithmus erzielte Verbesserung der logarithmierten ML um einen Faktor von etwa 2^{150} ist zwar beachtlich, allerdings benötigen die dazu erforderlichen Berechnungen sehr viel Zeit. Auf einem aktuellen Desktop-PC (Intel Pentium P4 3,4GHz, 2048MB RAM) wird dazu ca. 1,5 Wochen gerechnet.

Abbildung 3.2 zeigt den Verlauf der logarithmierten Conditional Maximum Likelihood über die Iterationen des Baum-Welch-Algorithmus. Ihr Wert wird — wie bereits er-

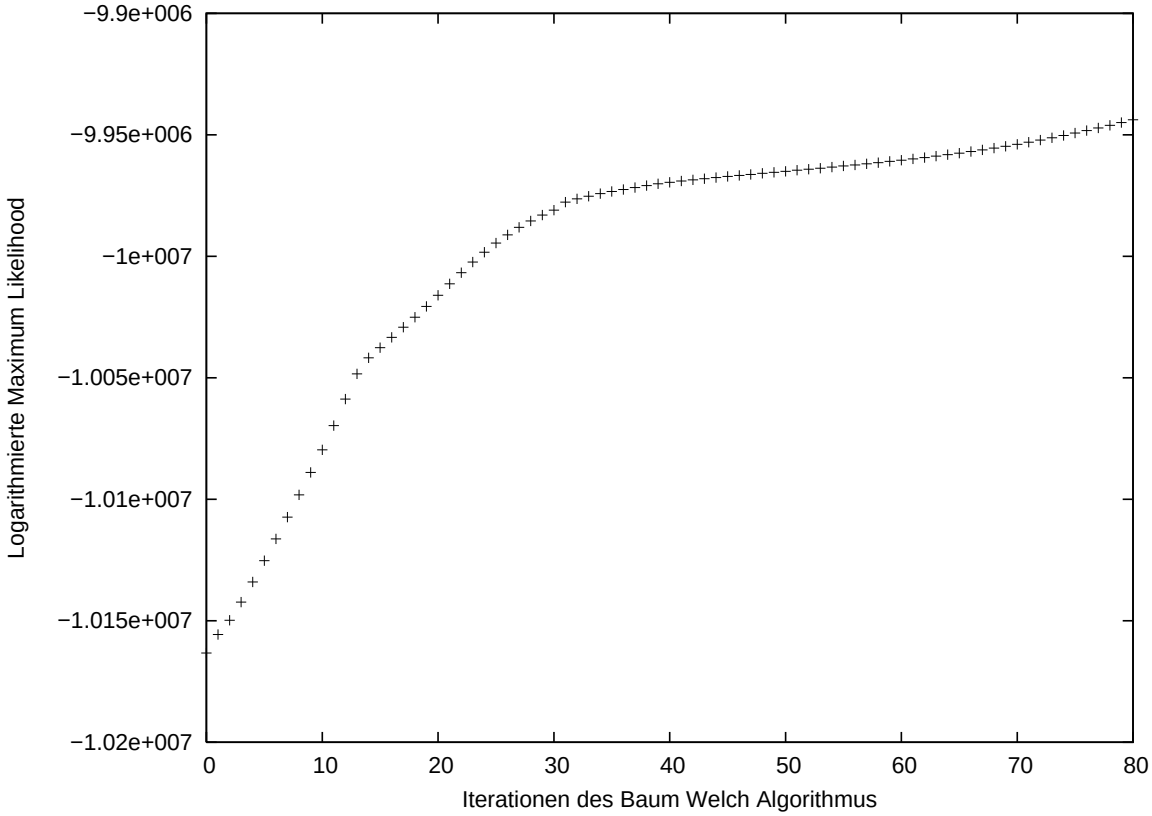


Abbildung 3.1: *Zeitliches Verhalten der logarithmierten ML beim Baum-Welch-Algorithmus.* Auf das zufällig generierte HMM werden 80 Iterationen des Baum-Welch-Algorithmus angewendet und nach jeder Iteration die logarithmierte ML aufgetragen. Diese steigt kontinuierlich an.

wähnt — nicht in jeder Iteration erhöht, sondern unterliegt Schwankungen. Eine Einordnung der Güte des aus der Anpassung durch den Baum-Welch-Algorithmus resultierenden HMMs wird in Kapitel 3.6 vorgenommen.

Unter Berücksichtigung der hohen Laufzeit jeder einzelnen Iteration des Baum-Welch-Algorithmus muss gesagt werden, dass ihr Nutzen, also der dadurch erreichte Gewinn an Genauigkeit der Vorhersagen, eher gering ist. Da die interessantere Information in diesem Fall die Markierung der Zustände, also letztendlich die Vorhersage der CpG-Inseln, ist, wäre es wünschenswert, die CML zu steigern. Dies leistet der Baum-Welch-Algorithmus leider nicht. Das Verhalten der ML ist eher zweitrangig. Ob die Anwendung von beispielsweise einer oder weniger Iterationen des Baum-Welch-Algorithmus auf die durch einen evolutionären Algorithmus generierten HMMs sinnvoll ist, muss noch überprüft werden. Die Rechenzeit könnte alternativ dazu verwendet werden, eine größere Anzahl an Individuen zu erzeugen und zu untersuchen.

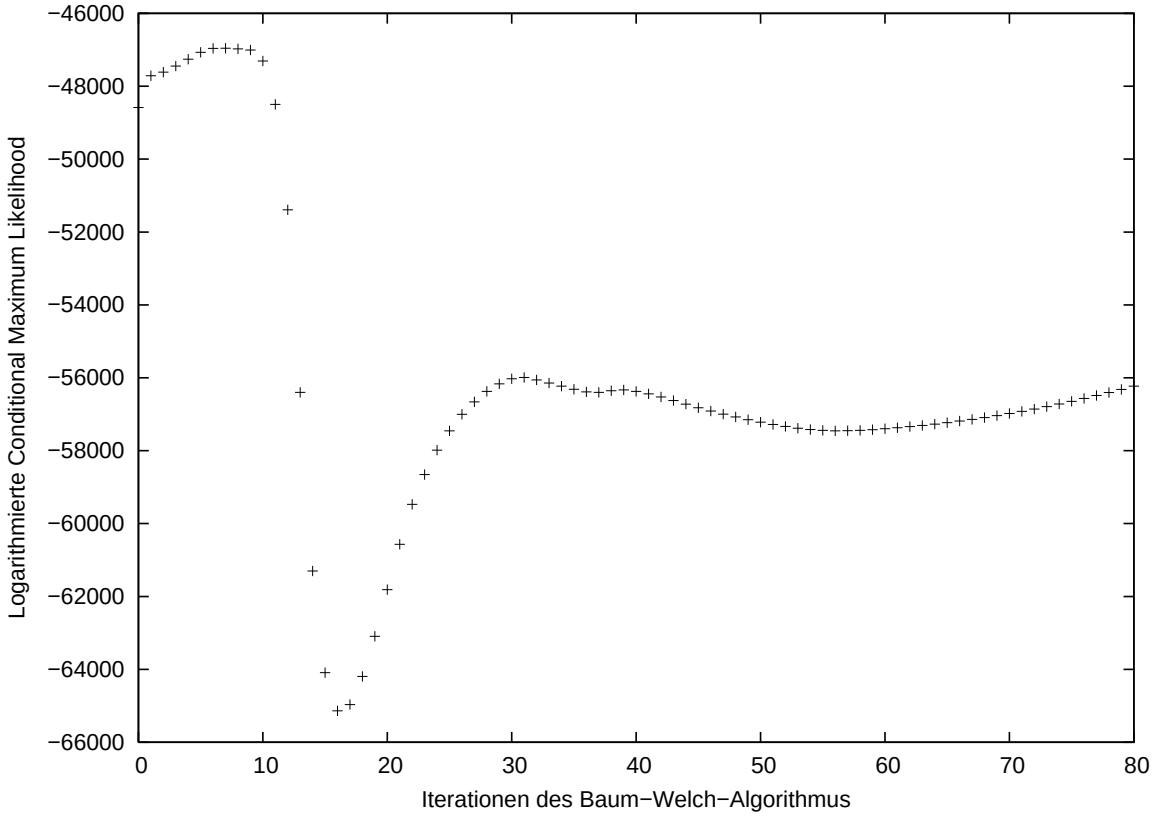


Abbildung 3.2: *Zeitliches Verhalten der logarithmierten CML beim Baum-Welch-Algorithmus.* Nach jeder der 80 Iterationen des Baum-Welch-Algorithmus wird die logarithmierte CML aufgetragen. Im Gegensatz zur logarithmierten ML steigt diese nicht kontinuierlich an.

3.4 Bewertungsmöglichkeit von HMMs

Um die durch den Baum-Welch-Algorithmus erzeugten HMMs bewerten zu können, werden die folgenden Gütemaße eingeführt und berechnet. Diese sollen dabei helfen, die Empfindlichkeit eines HMMs auf CpG-Inseln und die Exaktheit der aus ihm hervorgehenden Markierungen zu beschreiben.

- **Nukleotid-Sensitivität**: Prozentualer Anteil der Nukleotide, die in CpG-Inseln liegen und die auch korrekt als solche vorhergesagt werden, also

$$\text{Sensitivität} = \frac{\text{Anzahl der richtig klassifizierten Positiven}}{\text{Anzahl der Positiven}} \cdot 100\% \tag{3.2}$$

- **Nukleotid-Spezifität**: Prozentualer Anteil der Nukleotide, die als in CpG-Inseln liegend vorhergesagt werden und auch tatsächlich in einer CpG-Insel liegen, d. h.

$$\text{Spezifität} = \frac{\text{Anzahl der richtig klassifizierten Positiven}}{\text{Anzahl der positiv Klassifizierten}} \cdot 100\% \tag{3.3}$$

- **Verpasste CpG-Inseln**: Prozentualer Anteil an realen CpG-Inseln, die nicht von einer vorhergesagten CpG-Insel überlappt werden
- **Falsche CpG-Inseln**: Prozentualer Anteil an vorhergesagten CpG-Inseln, die nicht von einer tatsächlichen CpG-Insel überlappt werden

Sensitivität:

Original	0	0	**1**	1	1	**1**	**1**	0	0	$\frac{3}{5} \cdot 100\% = 60\%$
Vorhersage	1	1	**1**	0	0	**1**	**1**	1	0	

Spezifität:

Original	0	0	**1**	1	1	**1**	**1**	0	0	$\frac{3}{6} \cdot 100\% = 50\%$
Vorhersage	1	1	**1**	0	0	**1**	**1**	1	0	

Abbildung 3.3: *Beispiel für der Berechnung der Sensitivität und Spezifität.* Die in Original und Vorhersage übereinstimmend als CpG-Inseln markierten Nukleotide sind mit einem fettgedruckten Rahmen gekennzeichnet. Ihre Anzahl macht sowohl bei der Sensitivität als auch bei der Spezifität den Zähler aus. Die in den Nenner eingehenden Felder sind grau hinterlegt. Für die Sensitivität werden dazu die 1-Markierungen in der Originalsequenz, für die Spezifitiät die 1-Markierungen in der Vorhersagesequenz gezählt.

Abbildung 3.3 visualisiert die Berechnung der Sensitivität und der Spezifititität anhand einer kurzen Beispielsequenz. Um die Gütemaße berechnen zu können, muss allerdings zunächst anhand des HMMs eine Markierungssequenz vorhergesagt werden. Wünschenswert wäre es, diejenige zu finden, die Gleichung 2.7 maximiert. Jeder Pfad durch das HMM erzeugt eine Markierungssequenz, jedoch können einer Markierungssequenz durchaus mehrere Pfade zugrunde liegen. Daher ist zur Zeit kein effizienter Algorithmus bekannt, der die Maximierung von Gleichung 2.7 auf Seite 16 garantiert. Eine Approximation des Ergebnisses wird durch den Viterbi-Algorithmus und anschließendes Ablesen der Markierungen zu den Zuständen auf dem wahrscheinlichsten Pfad erreicht.

Für die Ermittlung der Gütemaße wird im Folgenden immer die in Abschnitt 3.2 beschriebene Validierungsdatenmenge verwendet. Dies ist erforderlich, um eine Aussage darüber treffen zu können, wie die mithilfe der Trainingsdatenmenge angepassten HMMs auf unbekannten Daten operieren.

3.5 Alternative zum Viterbi-Algorithmus

In Krogh (1997) findet sich eine zweite Approximation für die Markierungssequenz, die die Gleichung 2.7 maximiert. Diese soll auf Kosten der Laufzeit bessere Werte erzielen. Der dort beschriebene Algorithmus verwendet Hypothesen h_i, die jeweils eine

mögliche Markierungssequenz für die Teilsequenz der Länge i vom Nukleotid x_1 bis x_i repräsentieren. Es sei $\text{Prob}_s(h_i)$ die Wahrscheinlichkeit der Ausgabe der Nukleotidsequenz $x_1 \dots x_i$ unter der Bedingung, dass nur Pfade der Länge i, die die Markierungen h_i erzeugen und in Zustand s enden, betrachtet werden.

Für die anfänglich möglichen Hypothesen, die nur aus einer einzelnen Markierung Y bestehen, werden die Wahrscheinlichkeiten $\text{Prob}_s(Y)$ durch eine einfache Fallunterscheidung ermittelt. Trägt der Zustand s die Markierung Y, entspricht sie der Wahrscheinlichkeit vom Startzustand s_0 in den Zustand s zu gelangen und daraufhin im Zustand s das erste Symbol x_1 der Sequenz auszugeben. Falls die Markierung von s nicht mit Y übereinstimmt, gilt $\text{Prob}_s(Y) = 0$. Formal bedeutet dies

$$\text{Prob}_s(Y) = \begin{cases} a(s_0, s) \cdot e_s(x_1) & \text{falls } label(s) = Y \\ 0 & \text{falls } label(s) \neq Y \end{cases} \tag{3.4}$$

Bei der hier betrachteten Problemstellung, dem Auffinden von CpG-Inseln, sind für Y nur die Markierungen 0 und 1 möglich.

Ausgehend von der Annahme, dass die Wahrscheinlichkeiten $\text{Prob}_s(h_i)$ für alle Hypothesen h_i der Länge i und alle $s \in \mathcal{S}\backslash\{s_0\}$ bekannt sind, können nun diese Werte für Hypothesen der Länge $i+1$ iterativ berechnet werden. Für die Hypothese h_iY, also dass auf die Markierungen aus h_i die Markierung Y folgt, ergibt sich $\text{Prob}_s(h_iY)$ wie folgt: Falls die Markierung von s nicht mit Y übereinstimmt, gilt $\text{Prob}_s(h_iY) = 0$, andernfalls gilt

$$\text{Prob}_s(h_iY) = \left(\sum_{s' \in \mathcal{S}\backslash s_0} \text{Prob}_{s'}(h_i) a_{s',s} \right) e_s(x_{i+1}).$$

Dieses Vorgehen hat allerdings den Nachteil, dass die Anzahl der zu berücksichtigenden Hypothesen exponentiell in der Länge der Hypothesen, also der Länge der Sequenz, ansteigt. Im Fall von CpG-Inseln werden für jede Hypothese der Länge i demnach zwei Hypothesen der Länge $i+1$ erzeugt, nämlich einerseits, dass an der $(i+1)$-ten Stelle der Markierungssequenz die Markierung 0 folgt und andererseits die Markierung 1. Es müssen also letztendlich 2^l mögliche Hypothesen der Länge l berücksichtigt werden, was für eine Symbolsequenz von lediglich 1000 Nukleotiden bereits mehr als 10^{300} Hypothesen bedeuten würde.

Da dies aufgrund der Laufzeit und des Platzbedarfs nicht praktikabel ist, wird eine Approximation der Lösung vorgeschlagen. Die Grundidee ist es, die Anzahl betrachteter Hypothesen stark auszudünnen. Für jede Hypothesenlänge i und jeden Zustand s wird nur die beste Hypothese $h_s(i)$, also die mit dem maximalen $\text{Prob}_s(h_i)$-Wert, berücksichtigt. Ähnlich wie bei dem One-Best-Algorithmus aus Schwartz und Chow (1990) wird davon ausgegangen, dass dies eine gute Approximation darstellt. Alle Hypothesen, die in keinem Zustand die höchste Wahrscheinlichkeit erreicht haben, werden

demnach nicht weiter berücksichtigt. Dies senkt die Anzahl der Hypothesen drastisch auf $n \cdot l$. Die Hypothese der Länge l mit der höchsten Wahrscheinlichkeit bildet damit die resultierende Markierungssequenz.

Algorithmus 10 Dekodierungsalgorithmus von Anders Krogh

Eingabe: Symbolsequenz $x = x_1 x_2 \dots x_l$

1: **for all** $s \in \mathcal{S} \backslash s_0$ **do**
2: **for** $Y = 0$ and 1 **do**
3: $\text{Prob}_s(Y) \leftarrow \begin{cases} a(s_0, s) \cdot e_s(x_1) & \text{falls } label(s) = Y \\ 0 & \text{falls } label(s) \neq Y \end{cases}$
4: **end for**
5: **end for**
6: **for** $i = 1, \dots, l-1$ **do**
7: **for all** h_i **do**
8: **for** $Y = 0$ and 1 **do**
9: $\forall s \in \mathcal{S} \backslash s_0$:
10: $\text{Prob}_s(h_i Y) \leftarrow \begin{cases} \Big(\sum\limits_{s' \in \mathcal{S} \backslash s_0} \text{Prob}_{s'}(h_i) a_{s',s} \Big) e_s(x_{i+1}) & \text{falls } Y = \text{label}(s) \\ 0 & \text{falls } Y \neq \text{label}(s) \end{cases}$
11: **end for**
12: **end for**
13: **for all** $s \in \mathcal{S} \backslash \{s_0\}$ **do**
14: Verwerfe alle Einträge bis auf das Maximum $\max\limits_{h_{i+1}} \{\text{Prob}_s(h_{i+1})\}$
15: **end for**
16: **end for**
17: Wähle Hypothese h_l mit maximalem Wert $\max\limits_{h_l} \left\{ \sum\limits_s \text{Prob}_s(h_l) \right\}$

Der komplette Ablauf wird in Algorithmus 10 formal zusammengefasst. Der Zeitaufwand für diesen Algorithmus ist signifikant höher als beim Viterbi-Algorithmus. Daher stellt sich die Frage, ob die Qualität des Ergebnisses den Mehraufwand rechtfertigt. Im Folgenden wird nun überprüft, ob es einen Vorteil darstellt, diesen Algorithmus anstelle des Viterbi-Algorithmus für die Vorhersage der Markierungssequenzen eines schon fertig generierten HMMs zu verwenden. Dazu werden die in Abschnitt 3.4 eingeführten Gütemaße verwendet.

3.6 Entwicklung der Spezifität und der Sensitivität

Um nun die Güte der vom Baum-Welch-Algorithmus erzeugten HMMs einschätzen zu können, werden nach jeder zehnten Iteration des Algorithmus für das aktuelle HMM die Sensitivität und die Spezifität berechnet. Dies geschieht jeweils einmal unter Einbe-

ziehung der durch den Viterbi-Algorithmus generierten Markierungssequenz und einmal für die Markierungssequenz, die der Krogh-Algorithmus erzeugt hat. Zur Berechnung werden dabei die Validierungsdaten verwendet, die nicht im Zusammenhang mit dem Baum-Welch-Algorithmus verwendet wurden. Die Ergebnisse dieser Berechnungen sind in Abbildung 3.4 zusammengefasst.

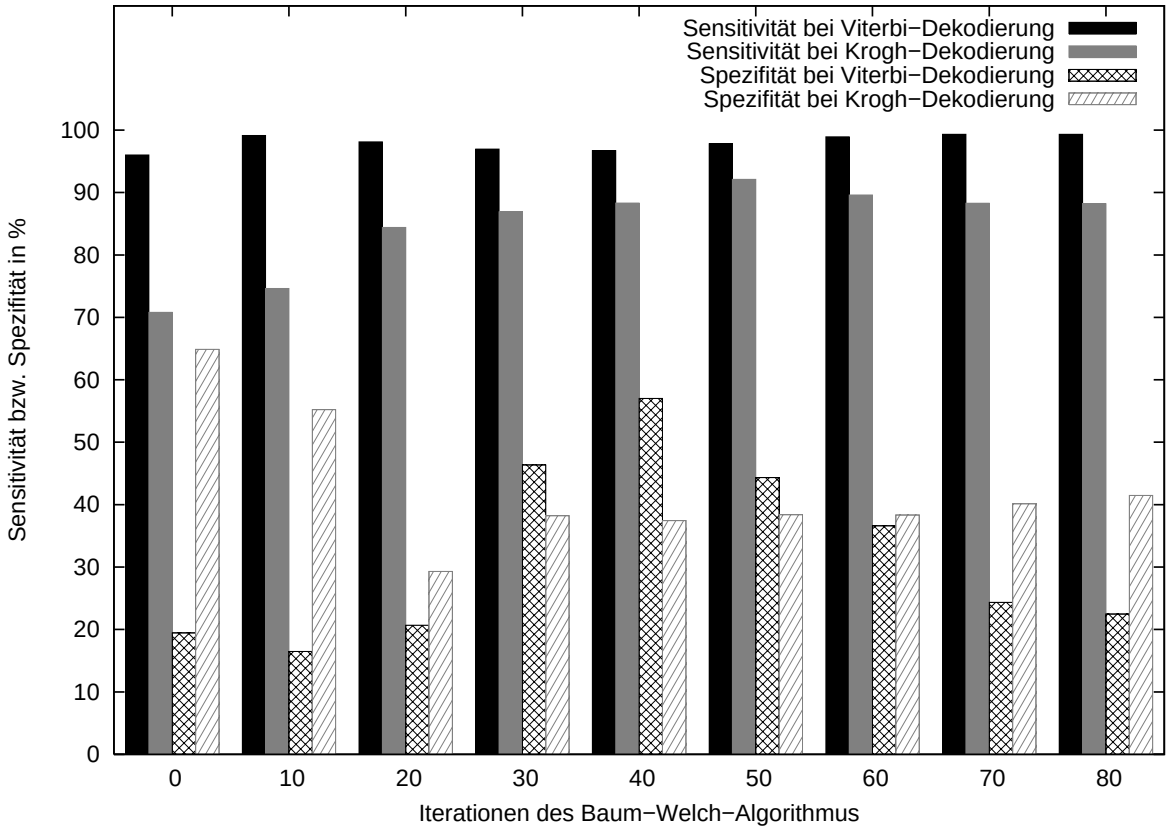


Abbildung 3.4: *Sensitivität und Spezifität im Verlauf des Baum-Welch-Algorithmus für Viterbi- und Krogh-Dekodierung.* Nach jeder zehnten Iteration werden für das aktuelle HMM die Sensitivität und die Spezifität jeweils einmal basierend auf der Viterbi- und Krogh-Dekodierung bestimmt.

Dabei fällt auf, dass für die Viterbi-Dekodierung die Sensitivität sehr hoch ist und zuletzt bei annähernd 100% liegt. Es werden also nahezu alle in CpG-Inseln gelegenen Nukleotide auch als solche erkannt. Für die Krogh-Dekodierung ist dieser Wert etwas geringer. Dies spiegelt sich auch in Tabelle 3.1 wider, wo die prozentualen Anteile an verpassten CpG-Inseln aufgeführt sind. Dort zeigt sich, dass bei der Viterbi-Dekodierung etwas weniger tatsächliche CpG-Inseln nicht überlappt werden als bei der Krogh-Dekodierung. Allerdings schwankt dieser Anteil verpasster CpG-Inseln bei der Viterbi-Dekodierung sehr stark, während er bei der Verwendung des Krogh-Algorithmus auf einem relativ niedrigen Niveau annähernd konstant bleibt.

Iterationen	Viterbi-Dekodierung in %	Krogh-Dekodierung in %
0	1,27	24,05
10	0	22,78
20	60,36	12,66
30	1,27	10,12
40	27,42	8,86
50	0	5,06
60	45,12	7,59
70	0	8,86
80	56,31	8,86

Tabelle 3.1: *Verpasste CpG-Inseln im Verlauf des Baum-Welch-Algorithmus für die beiden Dekodierungsvarianten.* Der Anteil der verpassten CpG-Inseln wird nach jeder zehnten Iteration basierend auf dem aktuellen HMM sowohl mit der Viterbi- als auch mit der Krogh-Dekodierung berechnet.

Iterationen	Viterbi-Dekodierung in %	Krogh-Dekodierung in %
0	68,31	22,45
10	74,72	35
20	75,57	46,15
30	30,16	38,24
40	93,66	38,81
50	37,50	38,57
60	88,72	41,42
70	55,45	40,57
80	80,08	41,42

Tabelle 3.2: *Falsche CpG-Inseln im Verlauf des Baum-Welch-Algorithmus für die beiden Dekodierungsvarianten.* Nach jeder zehnten Iteration wird für das aktuelle HMM der Anteil der falschen CpG-Inseln bei Viterbi- und Krogh-Dekodierung ermittelt.

Andererseits ist bei der Krogh-Dekodierung die Spezifität höher. Es werden also weniger Nukleotide fälschlicherweise als zu einer CpG-Insel gehörend markiert. Dies fällt ebenfalls in Tabelle 3.2 auf, wo die Anzahlen der falschen CpG-Inseln, also diejenigen markierten CpG-Inseln, die nicht von einer tatsächlichen CpG-Insel überlappt werden, aufgeführt sind. Hier werden bei der Krogh-Dekodierung wesentlich weniger falsche CpG-Inseln markiert.

Obwohl die Werte für die Sensitivität und Spezifität recht hoch liegen, zeichnet sich insgesamt jedoch im Verlauf des Baum-Welch-Algorithmus keine klare Tendenz für die

ermittelten Werte ab. Lediglich die Sensitivität steigt etwas an, bzw. bleibt annähernd konstant, während die Spezifität, sowie die Anzahlen der verpassten und falschen CpG-Inseln auch im Verlauf des Baum-Welch-Algorithmus noch stark schwanken. Hier zeichnet sich also keine deutliche Verbesserung durch die Durchführung einzelner Iterationen des Baum-Welch-Algorithmus ab.

Zusammenfassend lässt sich feststellen, dass die Dekodierungsvarianten mittels Viterbi- und Krogh-Algorithmus beide ihre Vor- und Nachteile aufweisen. Die Vorhersagen der Viterbi-Dekodierung besitzen die höhere Spezifität. Hier werden mehr Nukleotide als zu CpG-Inseln gehörend markiert. Dadurch werden fast alle in CpG-Inseln liegenden Nukleotide abgedeckt. Gleichzeitig verursacht dies eine meist etwas geringere Spezifität. Die Vorhersagen der Krogh-Dekodierung hingegen decken etwas weniger in CpG-Inseln liegende Nukleotide ab, erreichen zugleich jedoch eine höhere Spezifität. Zudem werden weniger falsche CpG-Inseln vorhergesagt. Letztendlich hängt es von der Anwendung ab, welche Dekodierung gewählt werden sollte. In dem vorliegenden Fall bietet die Viterbi-Dekodierung durch die hohe Sensitivität die Möglichkeit, alle vorhergesagten Bereiche noch einmal näher zu betrachten und so keine interessanten Abschnitte unberücksichtigt zu lassen.

Kapitel 4

Evolutionäre Ansätze

4.1 Erste Implementierung

Es wird zunächst eine Implementierung eines evolutionären Algorithmus benötigt, die in der Lage ist, auf einer Repräsentation für Hidden Markov Modelle zu operieren. Außerdem müssen die Mutations- und Rekombinationsoperatoren sowie ein Selektionsoperator für diese Problemstellung spezifiziert und implementiert werden. Des Weiteren ist eine Fitnessfunktion erforderlich, die in der Lage ist, die einzelnen Individuen möglichst gut auf einen Fitnesswert abzubilden. Dieser soll beschreiben, inwiefern das Individuum dazu geeignet ist, zu einer gegebenen Sequenz von Ausgabesymbolen eine Vorhersage über die Markierungen der Ausgabesymbole zu treffen.

4.1.1 Generierung der initialen Population

Die anfängliche Population wird erzeugt, indem für jedes ihrer Individuen zunächst die Anzahl der Zustände des HMMs festgelegt wird. Dazu wird gleichverteilt eine Zahl zwischen einer oberen und unteren Grenze gewählt. Die untere Grenze beträgt drei, da zusätzlich zum Startzustand zu jeder Markierung mindestens ein Zustand existieren muss. Die obere Grenze ist variabel, wird aber vorerst auf 11 gesetzt. Daher besteht die Möglichkeit, durch eine Veränderung der maximalen Zustandszahl deren Einfluss auf die Güte der erzeugten HMMs zu untersuchen. Die Anzahl der Ausgabesymbole ist bereits durch die Problemstellung festgelegt, im Fall von CpG-Inseln gibt es also immer vier Ausgabesymbole A, C, G und T.

Als nächstes müssen für die Zustände die dazugehörigen Markierungen festgelegt werden. Dies geschieht wiederum gleichverteilt, allerdings ist hinterher zu überprüfen, ob es zu jeder Markierung mindestens einen Zustand gibt und falls nicht, dies noch nachträglich einzufügen, indem eine mehrfach vorkommende Markierung in diese zuvor nicht vergebene Markierung geändert wird. Der Startzustand bekommt — wie bereits ausgeführt — zur einfacheren Handhabung in der Implementierung eine eigene Markierung, um ihn von den anderen Zuständen unterscheiden zu können.

Die Zustandsübergangs- und Ausgabewahrscheinlichkeiten werden gleichverteilt aus dem Intervall $[0, 1)$ bestimmt und anschießend normiert. Danach werden von den einzelnen Wahrscheinlichkeiten die Logarithmen bestimmt und abgespeichert, um die in Abschnitt 3.1.1 erläuterte Berechnung zu ermöglichen.

Wie viele Individuen eine Population umfassen soll, wird später mithilfe der sequentiellen Parameteroptimierung bestimmt. Da die Berechnung der Fitnesswerte oft sehr rechenintensiv ist, sind kleine Populationen in diesem Punkt von Vorteil. Für Rekombinationsoperatoren ist hingegen eine große Diversität innerhalb der Population im Allgemeinen vorteilhaft, was wiederum für eine große Population spricht. Hier muss ein Kompromiss gefunden werden.

4.1.2 Mutations- und Rekombinationsoperatoren

Die Mutationsoperatoren werden ähnlich zu den in Thomsen (2002) vorgestellten Operatoren gewählt. Sie werden im Folgenden genauer erläutert.

- `addState` fügt einen neuen Zustand zu dem bestehenden HMM hinzu und versieht diesen mit einer gleichverteilt zufällig ausgewählten Markierung. Die Zustandsübergangs- und Ausgabewahrscheinlichkeiten werden ebenfalls gleichverteilt zufällig gewählt. Um eine gültige Wahrscheinlichkeitsverteilung zu erhalten, wird eine Normierung durchgeführt.

- `deleteState` löscht — falls möglich — einen der Zustände des HMMs. Es muss dabei sichergestellt werden, dass weiterhin mindestens ein Zustand zu jeder Markierung existiert. Der Startzustand kann nicht gelöscht werden. Da durch das Löschen eines Zustandes sich die Wahrscheinlichkeit, in einen der verbleibenden Zustände zu gelangen, erhöht, ist auch hier eine Normierung nötig.

- `modifyStateLabel` ändert — falls möglich — die Markierung eines Zustandes. Dabei muss ebenfalls gewährleistet werden, dass mindestens ein Zustand zu jeder Markierung bestehen bleibt, die Markierung des Startzustands kann nicht verändert werden.

- `addTransition` erzeugt einen Übergang zwischen zwei Zuständen, indem ein Zustandsübergang, der zuvor die Wahrscheinlichkeit 0 hatte, auf einen gleichverteilt zufälligen Wert aus $(0, 1)$ gesetzt wird. Es muss anschließend normiert werden.

- `deleteTransition` löscht einen Zustandsübergang, indem die Wahrscheinlichkeit für diesen Übergang auf 0 gesetzt wird und normiert daraufhin die Wahrscheinlichkeiten.

- `swapAEntry` tauscht zwei zufällig gewählte Einträge der Zustandsübergangsmatrix gegeneinander aus. Hier ist ebenfalls eine Normierung notwendig.

- `swapEEntry` vertauscht zwei zufällig gewählte Einträge der Emissionsmatrix und normiert das Ergebnis.

- `modifyAEntry` verändert einen zufällig gewählten Eintrag der Zustandsübergangsmatrix, indem zu dem alten Wert ein normalverteilter Wert mit Varianz $\frac{1}{\text{Generation}+1}$ und Erwartungswert 0 addiert wird. Abschließend ist eine Normierung nötig.

- `modifyEEntry` verändert einen zufällig gewählten Eintrag der Emissionsmatrix, indem zu dem alten Wert ein normalverteilter Wert mit Varianz $\frac{1}{\text{Generation}+1}$ und Erwartungswert 0 addiert wird und führt eine Normierung der Wahrscheinlichkeiten durch.

Um ein neues Individuum zu generieren, wird zunächst einer der Mutationsoperatoren angewendet. Damit jedoch nicht nur kleine Veränderungen der Individuen, sondern auch globale Schritte durch den Suchraum ermöglicht werden, wird mittels einer geometrischen Verteilung bestimmt, wie viele zusätzliche Mutationsschritte sequentiell auf das Individuum angewendet werden sollen. Sei Prob(Mutation) der Parameter der geometrischen Verteilung. Somit werden mit der Wahrscheinlichkeit $\text{Prob}(X = n) = \text{Prob}(\text{Mutation})^n \cdot (1 - \text{Prob}(\text{Mutation}))$ noch n zusätzliche Mutationsschritte durchgeführt. Die Wahrscheinlichkeit Prob(Mutation) soll ebenfalls durch sequentielle Parameteroptimierung angepasst werden. Welcher Mutationsoperator jeweils in den einzelnen Schritten gewählt wird, wird gleichverteilt zufällig gewählt. Damit hat jeder einzelne Mutationsoperator eine Wahrscheinlichkeit von $\frac{1}{9}$ gewählt zu werden.

Da die in Thomsen (2002) verwendeten Rekombinationsoperatoren dort nicht näher beschrieben werden, werden hier die im Folgenden erläuterten beiden Operatoren eingesetzt.

Die Rekombination auf der Emissionsmatrix erfolgt, indem zufällig zwei Crossoverpunkte bestimmt werden. Seien n_1 bzw. n_2 die Zustandsanzahlen der Individuen. Damit betragen die Größen der Emissionsmatrizen $n_1 \times m$ bzw. $n_2 \times m$. Die zwei Crossoverpunkte werden so bestimmt, dass die dazwischen liegenden Zeilen in beiden Matrizen noch enthalten sind. Daher werden sie aus $\{1, 2, \ldots, \min\{n_1, n_2\}\}$ gewählt. Zwischen den Crossoverpunkten werden dann die kompletten Zeilen der Emissionsmatrizen ausgetauscht. Dies ist möglich, da die Breite der Matrix durch die Anzahl m der Ausgabesymbole bestimmt und daher bei allen Individuen gleich ist. Abbildung 4.1 skizziert dieses Vorgehen.

Da die Breite der Transitionsmatrix von der Anzahl der Zustände im jeweiligen HMM abhängt, muss die Rekombination dort etwas anders ablaufen als bei der Emissionsmatrix. Es können nicht die kompletten Zeilen zwischen den Crossoverpunkten ausgetauscht werden, da sie unterschiedlich lang sein können. Es soll zwischen der $n_1 \times n_1$ Matrix A und der $n_2 \times n_2$ Matrix B ein Crossover durchgeführt werden, wobei o. B. d. A.

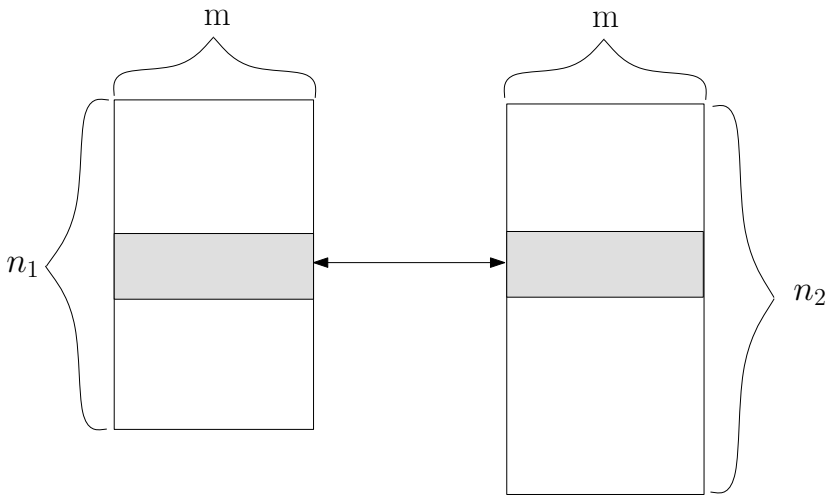


Abbildung 4.1: *Crossover auf den Emissionsmatrizen.* Zwischen den gleichverteilt zufällig aus $\{1, 2, \ldots, \min\{n_1, n_2\}\}$ gewählten Crossoverpunkten werden die kompletten Zeilen der Matrizen gegeneinander ausgetauscht.

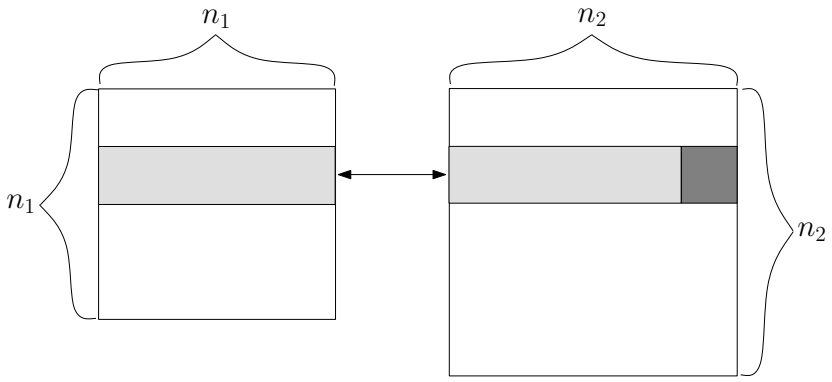


Abbildung 4.2: *Crossover auf den Transitionsmatrizen.* Durch die unterschiedlichen Zeilenlängen werden die Zeilen zwischen den gleichverteilt zufällig ermittelten Crossoverpunkten auf $\min\{n_1, n_2\}$, hier n_1, gekürzt und untereinander ausgetauscht. Dabei verfallen überschüssigen Elemente aus der $n_2 \times n_2$ Matrix und nach dem Austauschen der Zeilen fehlende Elemente werden gleichverteilt zufällig belegt. Abschließend erfolgt eine Normierung der veränderten Zeilen.

$n_1 \leq n_2$ gilt. Die beiden Crossoverpunkte müssen in $\{1, 2, \ldots, \min\{n_1, n_2\}\}$, also hier in $\{1, 2, \ldots, n_1\}$ liegen. Die Zeilen, die in A eingefügt werden sollen, werden auf die Länge n_1 gekürzt. Falls $n_1 \neq n_2$, verfallen dabei einige Einträge. Die Zeilen, die in B eingefügt werden sollen, werden an den Zeilenanfang gesetzt. Falls $n_1 \neq n_2$, fehlen am Ende der Zeilen noch einige Einträge, die dann zufällig belegt werden. Schließlich müssen die geänderten Zeilen in beiden Matrizen normiert werden, damit sie eine gültige Wahrscheinlichkeitsverteilung beschreiben. Die gesamte Operation ist in Abbildung 4.2 visualisiert.

Bei der Erzeugung eines Nachfolgeindividuums wird zusätzlich zur Mutation mit Wahrscheinlichkeit Prob(Rekombination) eine Rekombination durchgeführt. Wie dieser Wert belegt werden sollte, um besonders vorteilhafte Individuen zu erzeugen, soll im weiteren Verlauf mithilfe der sequentiellen Parameteroptimierung bestimmt werden. Welcher der beiden Operatoren in dem Falle einer Rekombination angewendet wird, wird wiederum zufällig bestimmt, wobei beide Operatoren mit der Wahrscheinlichkeit $\frac{1}{2}$ gewählt werden.

4.1.3 Erste Fitnessfunktion

In Thomsen (2002) wird als Fitnessfunktion

$$\Psi_1(\mathcal{I}_i) := \sum_{j=1}^{k} \log(\text{Prob}(x^j|\Theta)) + \omega \cdot \frac{\log(k+1)}{2} \cdot p_i \tag{4.1}$$

verwendet. Dabei bezeichnet ω den Balancierungsparameter. Die Anzahl p_i freier Parameter im Modell wird hauptsächlich durch die Zustandszahl bestimmt. Die Fitnessfunktion bezieht einerseits die Log-Likelihood zur Beschreibung der Vorhersagegenauigkeit und andererseits das Bayes'sche Informationskriterium zur Beschreibung der HMM-Komplexität mit ein.

4.1.4 Selektionsoperator

Zunächst werden die in Kapitel 2.2.5 beschriebenen Selektionsmethoden Rouletteradselektion und das stochastische universelle Sampling zur Selektion genutzt. Hierbei kommt es allerdings zu einem starken Verlust der Diversität. Die Fitnesswerte gemäß Gleichung 4.1 der einzelnen Individuen sind so unterschiedlich, dass nach dem Normieren der Werte nur noch ein Individuum mit einer Auswahlwahrscheinlichkeit von annähernd 1 übrig bleibt, während alle anderen Individuen mit einer Wahrscheinlichkeit von etwa 0 gewählt werden. Dies bewirkt, dass die Nachfolgegeneration häufig nur aus Kopien dieses einen Individuums besteht. Da dieser Effekt höchst unerwünscht ist, wird die Selektion abgewandelt: Jeweils nach der Auswahl eines Individuums wird dieses aus der Menge der noch selektierbaren Individuen entfernt; die Wahrscheinlichkeiten $\varphi(i)$ für die Auswahl eines der verbleibenden Individuen werden daraufhin gemäß Gleichung 2.8 neu berechnet.

Es sei noch einmal ausdrücklich erwähnt, dass sowohl die Individuen der Eltern-Generation, als auch die durch Mutation und Rekombination erzeugten Individuen zur Selektion herangezogen werden, es sich hier also um eine sogenannte Plus-Selektion handelt.

4.1.5 Zweite Fitnessfunktion

Neben den starken Unterschieden der Fitnesswerte hat die Fitnessfunktion aus Gleichung 4.1 ebenfalls den Nachteil, dass die Berechnung der einzelnen Fitnesswerte einen sehr hohen Rechenaufwand bedeutet. Dies wird durch die in der Summe versteckte Berechnung der CML verursacht. Es wäre wünschenswert, eine Alternative zu finden, die den Aufwand verringert und gleichzeitig weiterhin eine brauchbare Bewertung der Güte der Individuen hervorbringt.

Ein Lösungsansatz für diese Problematik bildet die Entwicklung einer neuen Fitnessfunktion. Jedes Individuum repräsentiert ein HMM. Um ein Individuum zu bewerten, werden dementsprechend mithilfe des Viterbi-Algorithmus die aus dem repräsentierten HMM folgenden Markierungen aller Trainingsbeispiele bestimmt. Der Fitnesswert berechnet sich anhand der prozentualen Übereinstimmungen mit den tatsächlichen Markierungen und anschließender Mittelwertbildung über alle k Trainingsbeispiele. Es ergibt sich also die Fitnessfunktion

$$\Psi_2(\mathcal{I}_i) := \frac{1}{k}\sum_{j=1}^{k}\frac{1}{l^j}\|\{x_i^j|1 \leq i \leq l^j \wedge y_i^j = label_{viterbi}(x_i^j)\}\| \cdot 100. \tag{4.2}$$

Dabei bezeichnet l^j die Länge der j-ten Trainingssequenz. Die Verwendung der prozentualen Übereinstimmung gewährleistet, dass jeder Trainingssequenz der gleiche Einfluss zukommt und die Fitness nicht durch einige lange Sequenzen dominiert wird.

Auf einen Term zur Berücksichtigung der Komplexität des durch das jeweilige Individuum repräsentierten HMMs wird an dieser Stelle bewusst verzichtet, um die Rechenzeit möglichst gering zu halten. Die HMMs können durch die Konstruktion der initialen Population und der Operatoren ohnehin nicht beliebig komplex werden, da die maximale Anzahl der Zustände von vorneherein auf 11 Zustände (inklusive Startzustand) begrenzt wurde.

Es wäre ebenso denkbar, anstelle des Viterbi-Algorithmus den Krogh-Algorithmus zur Berechnung der Fitnessfunktion zu verwenden. Allerdings ist hierbei der weitaus größere Zeitaufwand problematisch. Alternativ könnte der Krogh-Algorithmus zur Bestimmung der Markierungen für die durch den EA schon fertig angepassten HMMs einsetzt und geprüft werden, ob er dort einen Vorteil zum Viterbi-Algorithmus bietet. Dies wird in Kapitel 4.5 und 4.7.6 untersucht.

Als Selektionsoperator wurde weiterhin die abgewandelte Variante der Rouletteradselektion aus Kapitel 4.1.4 verwendet, da auch hier eine möglichst große Diversität der Nachfolgegeneration wünschenswert ist.

4.2 Sequentielle Parameteroptimierung

Die sequentielle Parameteroptimierung (SPO) stellt ein Verfahren dar, um gute Parameterbelegungen — beispielsweise für einen evolutionären Algorithmus — zu finden. In diesem Zusammenhang wird eine Belegung der Parameter des EAs auch Designpunkt genannt. Die SPO erzeugt zunächst eine Menge von initialen Designpunkten durch das Latin Hypercube Sampling (LHS). Dabei wird der n-dimensionale Parameterraum entlang jeder Achse in m gleich große Teile geteilt, die dann zusammen ein Gitter bilden. In diesem Gitter werden nun die gewünschten m Designpunkte so verteilt, dass in jeder achsenparallelen Hyperebene nur ein Designpunkt liegt. Dies bietet den Vorteil, dass nicht mehr Designpunkte benötigt werden, falls ein höherdimensionaler Parameterraum abgedeckt werden soll. In McKay u. a. (1979) finden sich ausführlichere Erläuterungen hierzu.

Nach der Generierung der ersten Designpunkte wird der evolutionäre Algorithmus mit diesen Parametereinstellungen durchgeführt und für jede Einstellung ein Funktionswert Y ermittelt, der die Güte der mit diesen Einstellungen erzielten Ergebnisse widerspiegelt. Da es sich bei einem evolutionären Algorithmus um einen randomisierten Algorithmus handelt, sind zuverlässige Aussagen erst nach mehreren Wiederholungen (Läufen) möglich. Für jeden Designpunkt wird daher eine vorgegebene Anzahl unabhängiger Wiederholungen mit unterschiedlichen Random-Seeds gestartet. Dies soll die Gefahr von zufällig guten oder schlechten Ergebnissen verringern. Daraufhin wird ein Regressionsmodell angewendet und so neue Designpunkte ermittelt, von denen erwartet wird, dass sie möglichst gute Ergebnisse liefern. Dieses Vorgehen wird sequentiell angewendet, um immer bessere Designpunkte zu finden (siehe Bartz–Beielstein 2006).

Die SPO-Toolbox (SPOT) für MATLAB bietet eine komfortable Möglichkeit der Anwendung dieser Methode. Dazu ist lediglich die Erstellung zweier Konfigurationsdateien nötig. In der einen (`.roi` für region of interest) wird festgelegt, welche Parameter angepasst werden sollen und in welchen Bereichen die Werte dieser Parameter verändert werden dürfen. In der zweiten (`.m`) werden verschiedene Einstellungen das Modell betreffend festgelegt. Hierzu zählen unter anderem die Anzahl von initialen Designpunkten für das Latin Hypercube Sampling und die Anzahl von Iterationen der SPO.

Für den in Kapitel 4.1 beschriebenen EA sollen nun die Populationsgröße sowie die Wahrscheinlichkeiten für Mutation und Rekombination angepasst werden. Dies bedeutet für die `.roi`-Datei, dass sie diese drei Einträge enthalten muss. Die Populationsgröße wird hier zwischen eins und zehn variiert, während für die Wahrscheinlichkeiten natürlich nur ein Wertebereich von 0 bis 1 möglich ist.

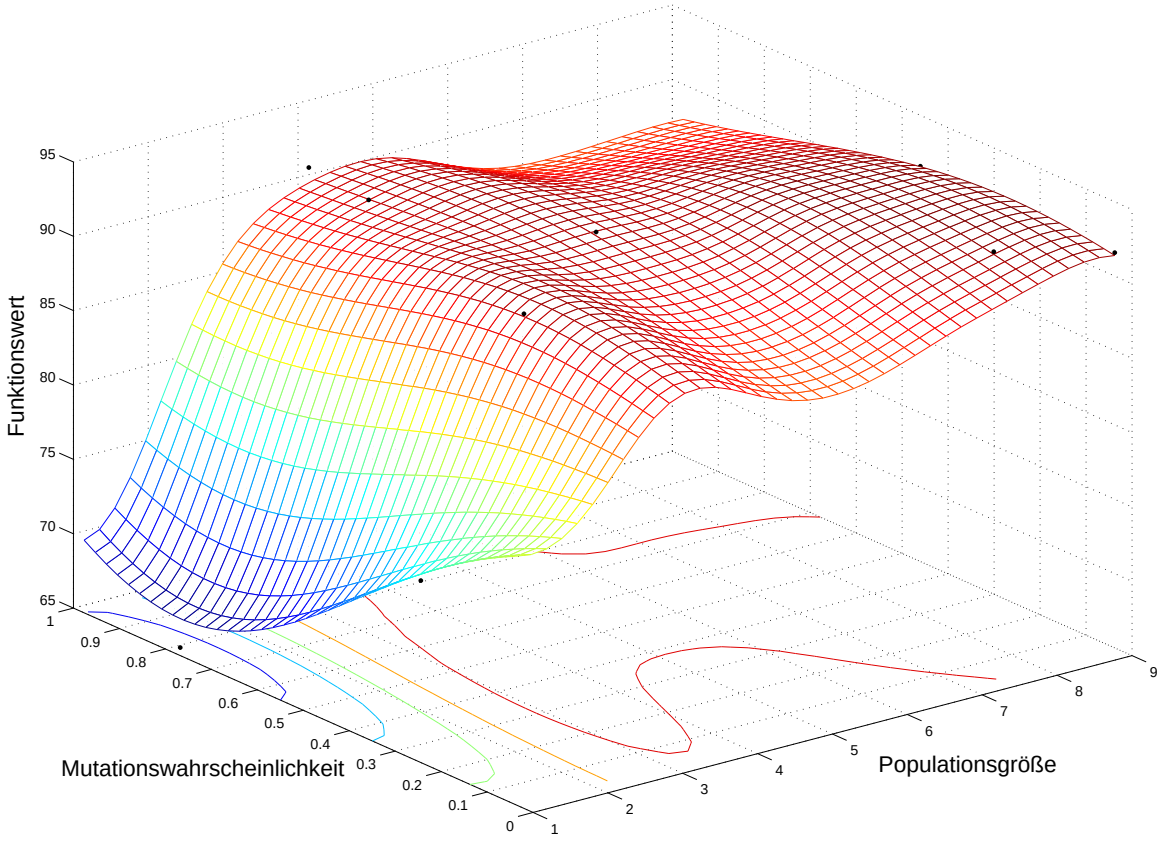


Abbildung 4.3: *Funktionswert Y in Abhängigkeit von der Populationsgröße und der Mutationswahrscheinlichkeit.* Das zugrunde liegende Design enthält 14 Designpunkte, für die jeweils zwei Wiederholungen mit verschiedenen Random-Seeds durchgeführt werden. Pro Wiederholung werden zehn Generationen des EAs mit der Fitnessfunktion Ψ_2 berechnet.

Es wird ein anisotropisches Modell gewählt und zunächst nur eine Iteration der SPO durchgeführt. Dazu werden 14 Designpunkte für das LHS gewählt. Aufgrund der hohen Laufzeit werden für jeden Designpunkt zwei Läufe mit unterschiedlichen Random-Seeds gestartet. Für jeden dieser Designpunkte werden zehn Generationen des evolutionären Algorithmus aus Kapitel 4.1 berechnet. Als Funktionswert Y, der die Güte des Designpunkts, also der Parametereinstellungen für den EA, beschreiben soll, dient hier das Maximum der Fitnesswerte, die bei der Evaluation der Individuen der zehnten und somit letzten Generation bestimmt werden. Mit dieser Konfiguration ergeben sich die in Abbildung 4.3 und 4.4 dargestellten Zusammenhänge.

Das von SPOT implementierte LHS verwendet rationale Zahlen, da es den vorgegebenen Wertebereich jeder Variablen in m äquidistante Rasterpunkte teilt. Für die Populationsgröße kommen allerdings offensichtlich nur natürliche Zahlen in Frage. Daher werden die von SPOT für die Populationsgröße festgelegten Werte jeweils gerundet.

Dadurch erklärt sich das Vorkommen von Einträgen bei fraktionalen Werten für die Populationsgröße in den folgenden Abbildungen.

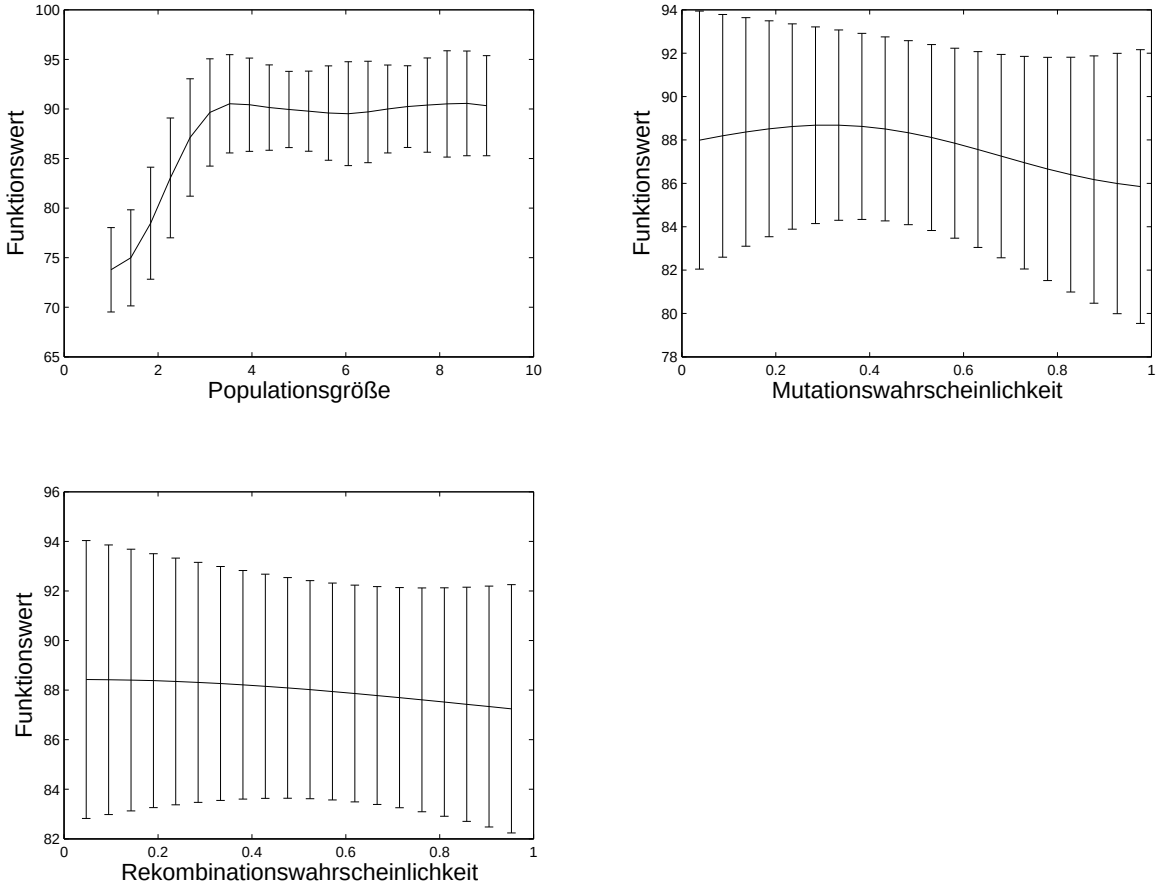


Abbildung 4.4: *Funktionswert Y in Abhängigkeit von den angepassten Parametern Populationsgröße, Mutationswahrscheinlichkeit und Rekombinationswahrscheinlichkeit.* Auch hier liegt das Design mit 14 Designpunkten, für die jeweils zehn Generationen mit je zwei verschiedenen Random-Seeds berechnet werden, zugrunde. Der Funktionswert für die Fitnessfunktion Ψ_2 zeigt sich kaum von der Rekombinationswahrscheinlichkeit beeinflusst.

In Abbildung 4.3 sowie in Abbildung 4.4 fällt auf, dass für sehr kleine Populationen mit drei oder weniger Individuen nur relativ geringe Funktionswerte erreicht werden. Diversität erweist sich hier als wichtig, jedoch nur in begrenztem Maße. Für Populationen mit vier bis zehn Individuen treten durchweg höhere, allerdings sehr ähnliche Funktionswerte auf. Zu beachten ist, dass bei steigender Populationsgröße auch der Zeitaufwand zur Berechnung proportional dazu ansteigt. Es sollte also eine Populationsgröße von vier bis fünf Individuen gewählt werden, da dies einen guten Kompromiss aus Fitness und Zeitaufwand darstellt.

Die Auswirkung der Mutationswahrscheinlichkeit auf die Funktionswerte erscheint wesentlich geringer als der Einfluss der Populationsgröße. Dennoch zeigt Abbildung 4.4, dass die Mutationswahrscheinlichkeit etwa im Bereich von 0,3 gewählt werden sollte, um möglichst gute Ergebnisse zu erzielen. Aus der in Abbildung 4.3 dargestellten Abhängigkeit des Funktionswertes vom Zusammenspiel aus Mutationswahrscheinlichkeit und Populationsgröße ergibt sich die Empfehlung, nur vier Individuen in einer Population zuzulassen, wenn eine Mutationswahrscheinlichkeit von 0,3 gewählt wird.

Den geringsten Einfluss auf den Funktionswert hat in den durchgeführten Läufen die Rekombinationswahrscheinlichkeit gezeigt. Die Änderung des Funktionswertes über die gesamte Variationsbreite der Rekombinationswahrscheinlichkeit ist sehr gering, während die Fehlerbalken wesentlich breiter sind. Die Diversität in der Population scheint demzufolge nicht für die Rekombination wichtig zu sein. Dies widerspricht der intuitiven Erklärung, dass speziell für die Rekombination eine Vielzahl von unterschiedlichen Individuen von Vorteil ist. Eine Aussage über eine besonders günstige Wahl der Rekombinationswahrscheinlichkeit ist also aufgrund der durchgeführten Tests nicht möglich.

4.3 Variation der Populationsgröße

Bei der SPO werden durch das dort verwendete LHS immer alle Parameter gleichzeitig verändert. Um nun den genauen Einfluss eines einzelnen Parameters betrachten zu können, werden zusätzliche Läufe bei gleichen Belegungen der übrigen Parameter benötigt. So sind die Auswirkungen eindeutig auf die einzige Veränderung zurückzuführen. Da die bisherigen Untersuchungen mithilfe der sequentiellen Parameteroptimierung ergeben haben, dass die Populationsgröße den stärksten Einfluss auf die Fitnesswerte zu haben scheint, werden im Folgenden die beiden verwendeten Extremwerte zwei und zehn für die Populationsgröße exemplarisch näher betrachtet. Hierzu werden für beide Werte jeweils fünf Läufe mit unterschiedlichen Random-Seeds gestartet, die Rekombinationswahrscheinlichkeit wird dabei konstant mit 0,5 und die Mutationswahrscheinlichkeit mit 0,3 gewählt.

Bei einer Populationsgröße von 10 werden in jeder Generation offensichtlich fünfmal so viele Individuen untersucht wie bei nur zwei Elternindividuen. Daher sollten bei der kleineren Population dementsprechend fünfmal so viele Generationen durchgeführt werden, um eine weitestgehende Chancengleichheit zu gewährleisten. Dies wird insbesondere dadurch klar, dass in der Zeit, die benötigt wird, um die Fitnesswerte für eine Generation von zehn Individuen zu berechnen, auch fünf Generationen von je zwei Individuen berechnet werden können. Da die Berechnung der Fitnesswerte den größten Rechenaufwand bedeutet, sollte hier die Rechenzeit, bzw. die Anzahl der bewerteten Individuen als Skala zugrunde liegen. Ob diese Zeit nun verwendet wurde, um die Fit-

nesswerte nur einer Generation mit vielen Individuen zu berechnen oder stattdessen mehrerer Generationen mit wenigen Individuen, macht dabei keinen Unterschied.

Aus diesem Grund werden jeweils 20 Generationen für die zwei Individuen große Population berechnet und je vier Generationen für die Populationsgröße zehn. Es werden jeweils n unabhängige Läufe durchgeführt. Daraufhin werden für jeden Lauf i und für jede Generation j das Maximum m_j^i der in dieser Generation des Laufs aufgetretenen Fitnesswerte bestimmt und das arithmetische Mittel $\overline{m}_j$ der m_j^i über die Läufe mit den verschiedenen Seeds ermittelt. Diese sind in Abbildung 4.5 aufgetragen, wobei die Fitnesswerte der initial generierten Population bei 0 aufgetragen sind und im weiteren Verlauf die Anzahl der während des Evolutionsprozesses bestimmten Fitnesswerte als Skala dient. Aufgrund der unterschiedlichen Populationsgrößen ergeben sich auch die verschiedenen Punkte, an denen die Werte bestimmt werden können.

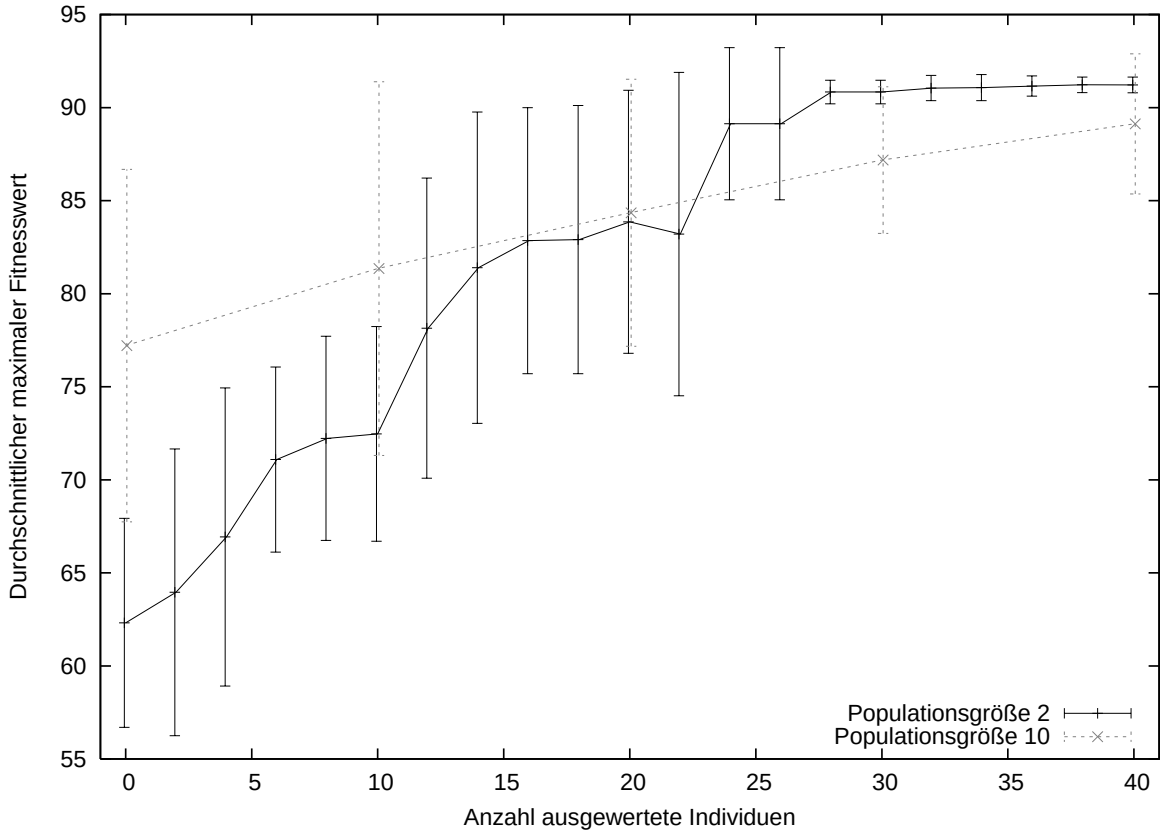


Abbildung 4.5: *Vergleich der Fitnesswerte bei den Populationsgrößen zwei und zehn.* Für beide Populationsgrößen werden fünf unabhängige Läufe durchgeführt und jeweils nach einer bestimmten Anzahl von ausgewerteten Individuen die dabei erreichten maximalen Fitnesswerte über diese Läufe gemittelt. Die Fehlerbalken ergeben sich aus der erwartungstreuen Stichprobenvarianz gemäß Gleichung 4.3.

Die in Abbildung 4.5 eingetragen Fehlerbalken ergeben sich aus der erwartungstreuen Stichprobenvarianz σ_j^2, die nach der Formel

$$\sigma_j = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\overline{m}_j - m_j^i)^2} \tag{4.3}$$

berechnet wird (siehe Fahrmeir u. a. 2003). In diesem Fall wurden jeweils fünf Läufe durchgeführt, es gilt also $n = 5$.

Auffällig ist, dass die initiale Population mit zehn Individuen einen wesentlich höheren maximalen Fitnesswert hat als bei zwei Individuen. Dies resultiert daraus, dass das Maximum der Fitnesswerte von zehn bzw. zwei zufällig generierten Individuen bestimmt wird und dieses erwartungsgemäß höher ist, wenn mehr Individuen zur Auswahl stehen. Bereits nach ca. 20 ausgewerteten Individuen hat der EA mit der kleineren Population diesen Nachteil jedoch ausgeglichen und erreicht deutlich schneller hohe Fitnesswerte mit ebenfalls geringeren Varianzen.

Während der oben beschriebenen Läufe wird, zusätzlich zu den Fitnesswerten, ebenfalls protokolliert, wann und wie oft mutiert und rekombiniert wird und welche Individuen letztendlich für die nächste Generation selektiert werden. Mit Schrittweite wird die zur Erzeugung eines Individuums durchgeführte Anzahl an Mutationen bezeichnet. Um herauszufinden, welche Schrittweiten bei den Mutationen auftreten und wie groß der jeweilige Anteil der erfolgreichen Mutationen ist, also wie viele dieser mutierten Individuen dann auch für die Population in der nächsten Generation selektiert werden, können allerdings nur die Individuen betrachtet werden, bei denen keine Rekombination durchgeführt wird. Dies liegt darin begründet, dass der Fitnesswert eines Individuums, für das zuvor Mutationsschritte und eine Rekombination durchgeführt wurden, durch beides beeinflusst wird und damit der Erfolg oder Misserfolg der durchgeführten Operationen nicht mehr klar zuzuordnen ist. In Abbildung 4.6 ist für den Fall einer Populationsgröße von zwei zu jeder Schrittweite aufgetragen, wie viele Individuen in den durchgeführten Läufen mutiert, aber gleichzeitig nicht Ergebnis einer Rekombination sind. Außerdem ist jeweils dargestellt, wie viele dieser Individuen nach der Durchführung der Mutationsschritte selektiert wurden, also in wie vielen der Fällen die Mutation „erfolgreich“ war. In Abbildung 4.7 sind die gleichen Daten für die Läufe mit Populationsgröße zehn dargestellt.

Hieraus lässt sich ablesen, dass — unabhängig von der Populationsgröße — sowohl Individuen mit großen als auch mit kleinen Schrittweiten den Sprung in die Nachfolgepopulation schaffen. Es werden öfter Individuen mit kleinen Schrittweiten erzeugt als Individuen mit großen Schrittweiten, was daraus resultiert, dass die Anzahl der Mutationsschritte, die für ein Individuen durchgeführt werden sollen, durch eine geometrische Verteilung bestimmt wird. Das Verhältnis von durchgeführten Mutationen zu erfolgreichen Mutationen ist nahezu gleich. Dies fällt insbesondere in dem Bereich

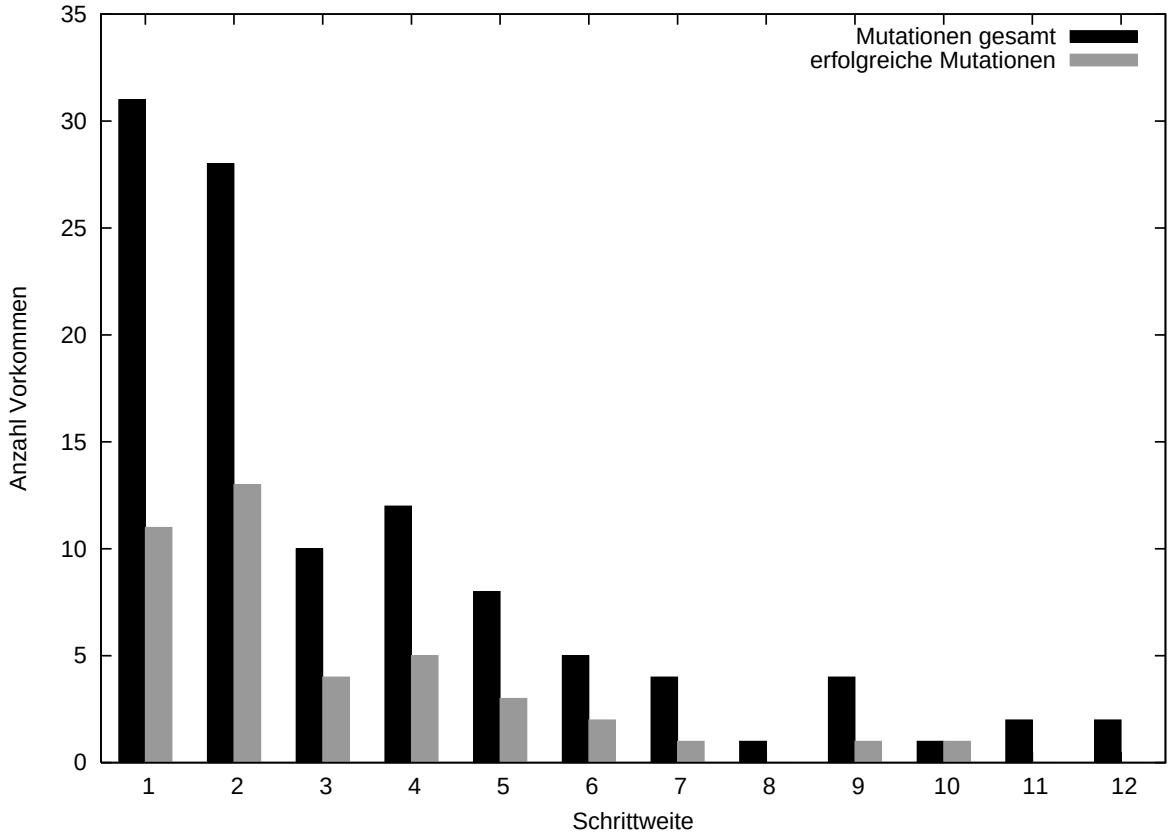


Abbildung 4.6: *Mutationen mit verschiedenen Schrittweiten bei einer Populationsgröße von zwei.* Hierbei werden in den fünf unabhängigen Läufen der Populationsgröße zwei die Vorkommen der Schrittweiten nur in den Fällen berücksichtigt, in denen keine Rekombination erfolgt. Die Gesamtzahl an Vorkommen der Schrittweiten ist den erfolgreichen Mutationen der jeweiligen Schrittweite gegenübergestellt.

geringer Schrittweiten auf. Letztendlich zeigt sich keine klare Tendenz der Abhängigkeit des Mutationserfolges von großen oder kleinen Schrittweiten.

4.4 Erhöhung der maximalen Zustandsanzahl

Bei den durchgeführten Läufen fällt besonders auf, dass mit der Fitnessfunktion aus Gleichung 4.2 nie Fitnesswerte über 91,58 erreicht werden. Dies entspricht einer prozentualen Übereinstimmung der vorhergesagten Markierungen mit den tatsächlichen Markierungen von durchschnittlich 91,58%. Hierbei ist allerdings die Anzahl Zustände für ein HMM auf zehn Zustände plus Startzustand beschränkt worden, was eine mögliche Ursache für die Stagnation des Fitnesswertes sein könnte.

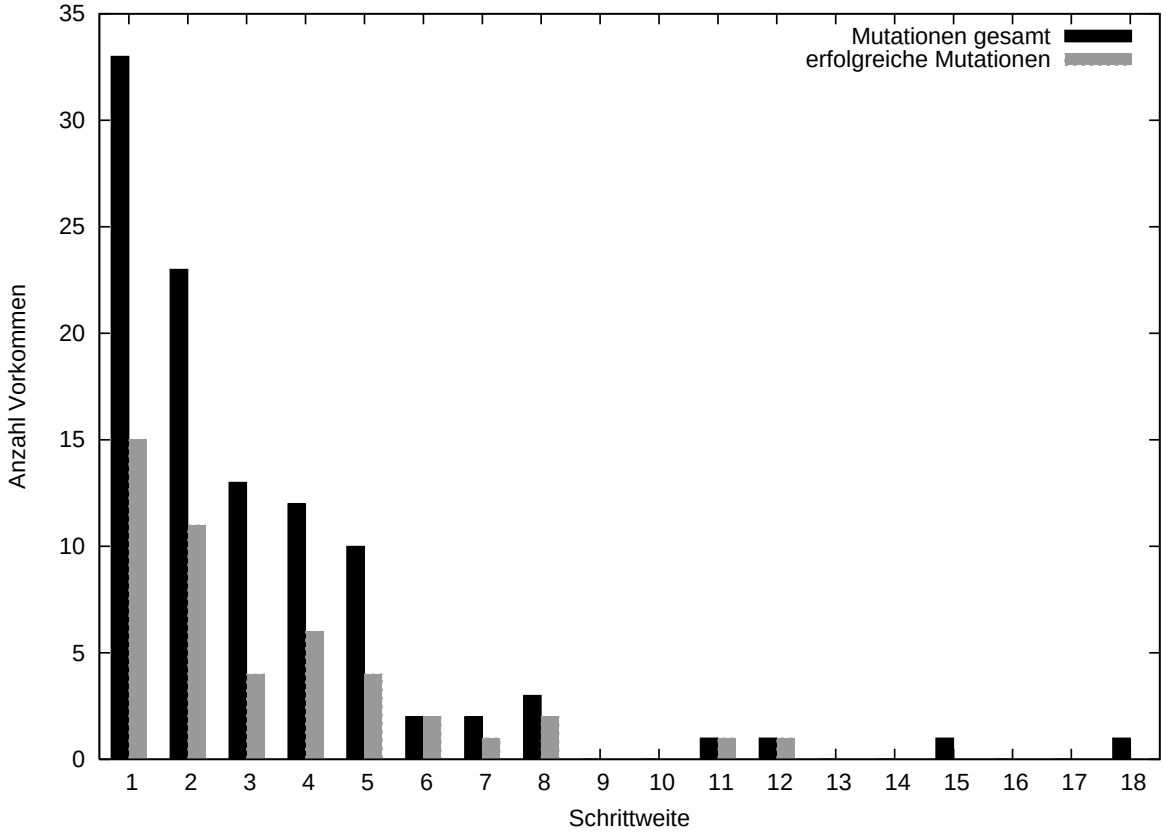


Abbildung 4.7: *Mutationen mit verschiedenen Schrittweiten bei einer Populationsgröße von zehn.* Hierbei werden in den fünf unabhängigen Läufen der Populationsgröße zehn die Vorkommen der Schrittweiten nur in den Fällen berücksichtigt, in denen keine Rekombination erfolgt. Die Gesamtzahl an Vorkommen der Schrittweiten ist den erfolgreichen Mutationen dieser Schrittweite gegenüber gestellt.

Zur Überprüfung dieser Hypothese wird die Zustandsanzahl nun von zehn auf 20 bzw. 30 Zustände plus Startzustand erhöht. Es werden für alle drei Beschränkungen der Zustandsanzahl der HMMs jeweils 100 Generationen mit ansonsten in allen Fällen identischer Konfiguration des EA berechnet. Die Populationsgröße wird auf zwei gesetzt, die Mutations- bzw. Rekombinationswahrscheinlichkeit werden auf 0,3 bzw. 0,5 festgelegt. Die Ergebnisse der Läufe sind in Abbildung 4.8 dargestellt.

Es zeigt sich, dass auch eine Erhöhung auf 30 Zustände plus Startzustand keine Erhöhung der maximalen Fitnesswerte zur Folge hat: Weiterhin ist der höchste erreichte Wert 91,58. Während bei HMMs mit maximal zehn Zuständen der maximale Fitnesswert ca. ab der 40. Generation weitestgehend konstant bleibt, treten bei 20 bzw. 30 Zuständen pro HMM bis ca. zur 80. Generation teils deutliche Schwankungen der Fitness auf. Dennoch liegen alle auftretenden maximalen Fitnesswerte unabhängig von

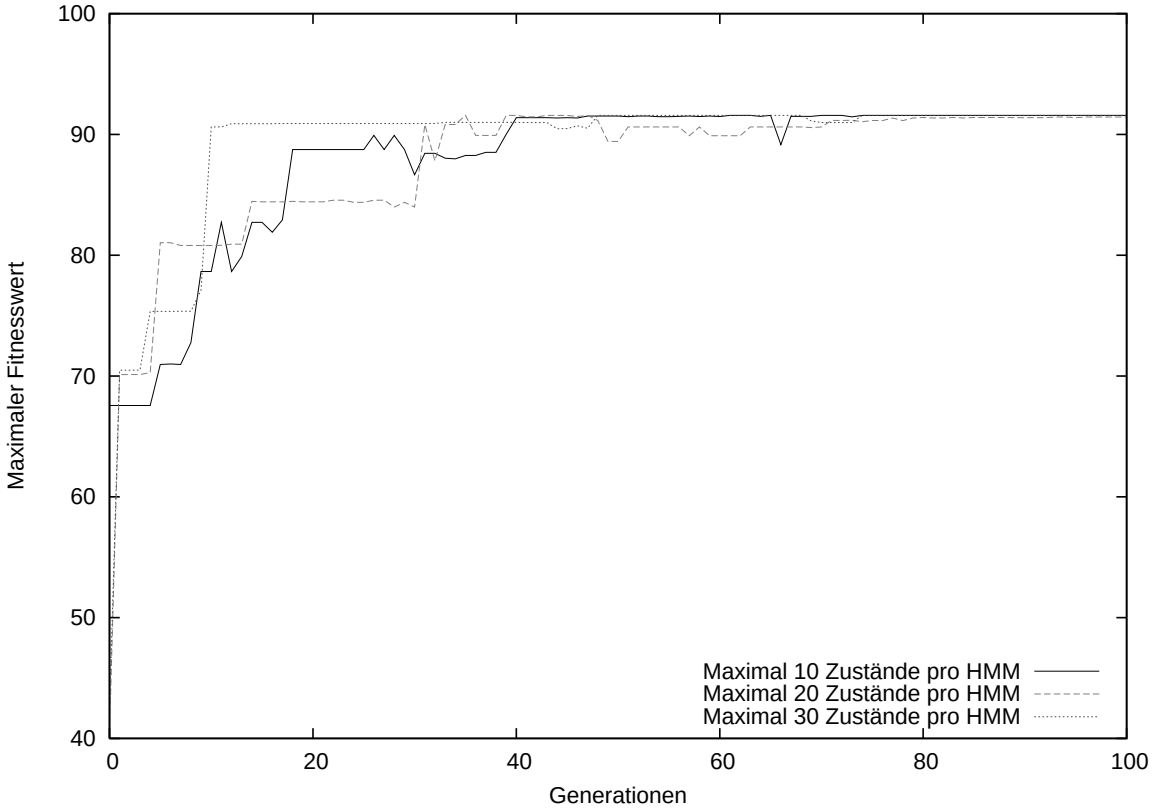


Abbildung 4.8: *Verhalten des Fitnesswertes bei unterschiedlichen Zustandsbeschränkungen der HMMs.* Für die maximalen Zustandsanzahlen von 10, 20 und 30 werden jeweils 100 Generationen mit ansonsten identischen Parametereinstellungen berechnet. Es ist jeweils der maximale Fitnesswert der jeweiligen Generation dargestellt.

der maximalen Zustandsanzahl des HMMs auf demselben Niveau. Dies spricht gegen die Hypothese, dass die limitierte Zustandsanzahl ursächlich für die Stagnation des Fitnesswertes ist. Zugleich erhöht sich der Rechenaufwand zur Fitnessauswertung mit steigender Zustandsanzahl. Somit wächst auch die Laufzeit des evolutionären Algorithmus. Aufgrund des mangelnden Vorteils dieser höheren Zustandsanzahl wird im weiteren Verlauf dieser Arbeit darauf verzichtet. Es muss also eine andere Erklärung für die Stagnation der Fitnesswerte geben.

4.5 Einschätzung der Güte der erzeugten HMMs

Die in Abschnitt 3.4 eingeführten Gütemaße bieten die Möglichkeit, die Güte der Ergebnisse des evolutionären Algorithmus unabhängig von der verwendeten Fitnessfunktion auf eine einheitliche Art zu bewerten und mit den Ergebnissen des Baum-Welch-

Algorithmus zu vergleichen. Daher werden für einige der durch den evolutionären Algorithmus generierten HMMs die Gütemaße ermittelt. Hierzu wird aus den, in Abschnitt 4.3 bereits beschriebenen, jeweils fünf unabhängigen, mit verschiedenen Random-Seeds durchgeführten Läufen zu den Populationsgrößen zwei und zehn jeweils das Ergebnis-HMM verwendet. Außerdem werden zusätzlich fünf unabhängige Läufe mit einer Populationsgröße von vier durchgeführt. Diese Populationsgröße hatte sich in der SPO als besonders vorteilhaft erwiesen. Bei der Populationsgröße von vier werden jeweils zehn Generationen berechnet, um genauso viele Individuen zu bewerten wie es bei den anderen beiden Populationsgrößen geschehen ist. Von diesen Läufen werden ebenfalls jeweils die Ergebnis-HMMs untersucht. Die Mutations- und Rekombinationswahrscheinlichkeiten wurden dabei — wie bereits erwähnt — auf 0,3 und 0,5 gesetzt. Zur Ermittlung der Werte werden wieder nur die Validierungsdaten und nicht die zur Fitnessberechnung herangezogenen Trainingsdaten verwendet.

Abbildung 4.9 zeigt die berechneten Sensitivitäts- und Spezifitätswerte für beide Dekodierungsvarianten. Hier zeigt sich die Viterbi-Dekodierung für die meisten der getesteten HMMs als vorteilhafter. Insgesamt sind jedoch die Werte für Sensitivität und Spezifität durchweg sehr niedrig.

Index	Populations-größe	Viterbi-Dekodierung in %	Krogh-Dekodierung in %
1	2	89,16	—
2	2	87,75	—
3	2	—	—
4	2	93,28	—
5	2	100	86,62
6	4	88,17	—
7	4	95,22	—
8	4	89,86	91,22
9	4	84,32	—
10	4	89,60	89,33
11	10	77,62	—
12	10	91,53	100
13	10	100	95,86
14	10	87,53	—
15	10	98,40	—

Tabelle 4.1: *Falsche CpG-Inseln im Verlauf des Baum-Welch-Algorithmus für die beiden Dekodierungsvarianten.* Hierbei handelt es sich um die Ergebnis-HMMs der je fünf unabhängigen Läufe mit den Populationsgrößen zwei, vier und zehn. Die Indices entsprechen denen aus Abbildung 4.9.

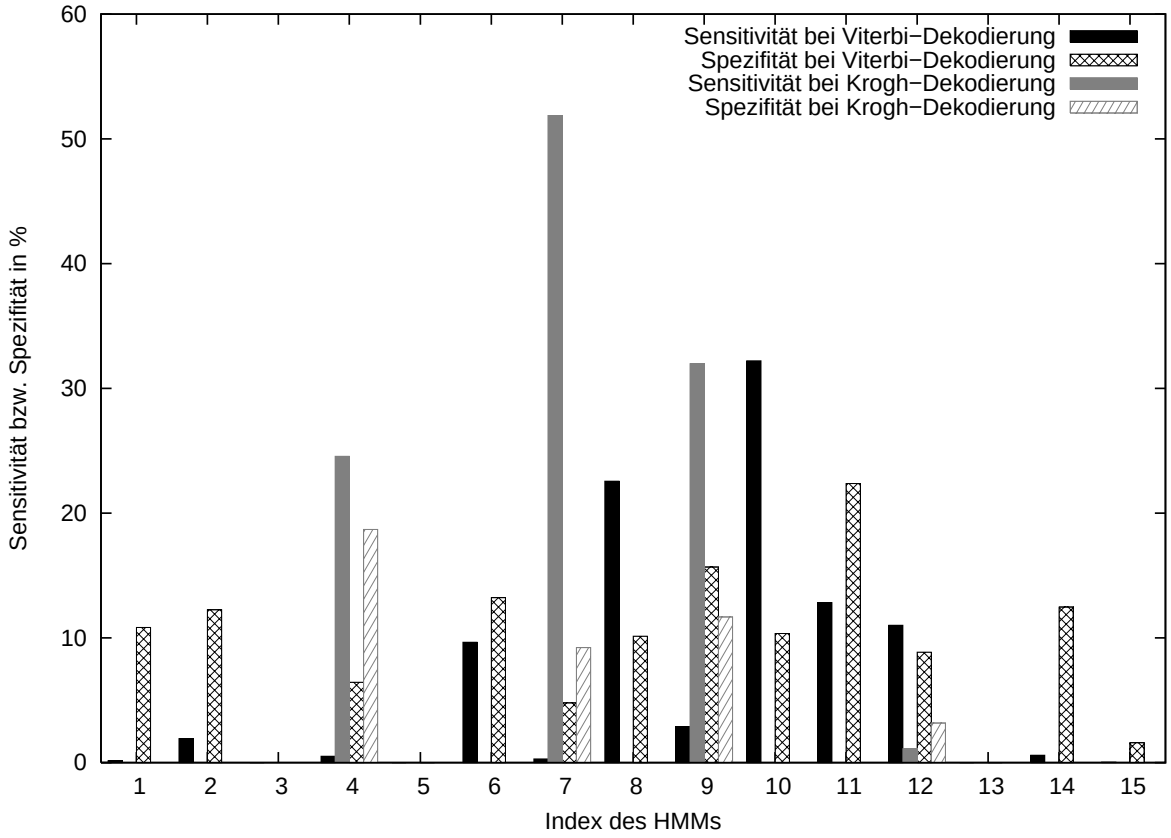


Abbildung 4.9: *Sensitivität und Spezifität für einige durch den EA erzeugte HMMs bei Viterbi- und Krogh-Dekodierung.* Für jede der drei Populationsgrößen werden je fünf unabhängige Läufe mit verschiedenen Random-Seeds und ansonsten gleicher Konfiguration durchgeführt. Für jeden der einzelnen Läufe sind für das Ergebnis-HMM die Sensitivität und die Spezifität aufgetragen. Die ersten fünf Indices beziehen sich auf die Läufe mit Populationsgröße zwei, die mittleren fünf mit Populationsgröße vier und die letzten fünf mit Populationsgröße zehn.

Für 10 der 15 getesteten HMMs wurden unter Verwendung der Krogh-Dekodierung — bei einem davon ebenfalls bei der Viterbi-Dekodierung — auf den Validierungsdaten keine Nukleotide als zu einer CpG-Insel gehörend markiert. Dies ist daran abzulesen, dass in den Fällen der prozentuale Anteil der falsch vorhergesagten CpG-Inseln sowie die Spezifität nicht berechenbar sind (in Tabelle 4.1 durch „—“ dargestellt). Außerdem ist für diese HMMs der prozentuale Anteil der verpassten CpG-Inseln 100% sowie die Sensitivität 0%.

Andererseits gibt es viele HMMs — insbesondere bei der Viterbi-Dekodierung — für die der Anteil an nicht gefundenen CpG-Inseln sehr gering ist. Oft werden sogar alle CpG-Inseln gefunden (siehe Tabelle 4.2). Jedoch ist in allen Fällen der Anteil an falsch vorhergesagten CpG-Inseln groß: Mehr als 3/4 der vorhergesagten CpG-Inseln werden

nicht von einer tatsächlichen CpG-Insel überlappt, wie Tabelle 4.1 zeigt. Des Weiteren ist für nahezu alle HMMs die Spezifität kleiner als 25%, d. h., dass mehr als 3/4 der Nukleotide, die als in CpG-Inseln liegend markiert wurden, nicht in einer solchen liegen, es sich also um falsche Vorhersagen handelt.

Insgesamt werden mit der Viterbi-Dekodierung etwas bessere Ergebnisse erzielt. Dies ist wahrscheinlich eine Folge dessen, dass diese für die Berechnung der Fitnesswerte verwendet wird. In diesem Fall stellt es also keinen Vorteil dar, für das durch den EA erzeugte HMM die Vorhersagen mit dem Krogh-Algorithmus zu treffen.

Index	Populations-größe	Viterbi-Dekodierung in %	Krogh-Dekodierung in %
1	2	54,43	100
2	2	0	100
3	2	100	100
4	2	18,99	100
5	2	100	6,32
6	4	0	100
7	4	0	100
8	4	0	0
9	4	26,58	100
10	4	0	0
11	10	0	100
12	10	0	100
13	10	100	40,50
14	10	20,25	100
15	10	84,81	100

Tabelle 4.2: *Verpasste CpG-Inseln im Verlauf des Baum-Welch-Algorithmus für die beiden Dekodierungsvarianten.* Für die Ergebnis-HMMs der jeweils fünf unabhängigen Läufe mit den Populationsgrößen zwei, vier und zehn wird der Anteil verpasster CpG-Inseln bestimmt. Die Indices entsprechen denen aus Abbildung 4.9.

Die Stagnation der Fitnesswerte bei 91,58 liegt demzufolge darin begründet, dass die Trainingsdaten nur sehr wenige CpG-Inseln enthalten und durch die gewählte Fitnessfunktion jedes Nukleotid gleich stark gewichtet wurde. Dies führt jedoch dazu, dass die Nukleotide außerhalb von CpG-Inseln den Fitnesswert dominieren und durch Markierung aller Nukleotide als nicht zu einer CpG-Insel gehörend schon eine sehr hohe Fitness erreicht wird, ohne dass das eigentliche Ziel — das Auffinden von CpG-Inseln — erfüllt wird. Dieser Problematik soll mit einer neuen Fitnessfunktion entgegengewirkt werden.

4.6 Dritte Fitnessfunktion

Da sich im letzten Abschnitt die Notwendigkeit einer besser auf die Problemstellung angepassten Fitnessfunktion zeigte, soll nun die Fitness basierend auf den Sensitivitäts- und Spezifitätswerten berechnet werden. Dies spiegelt den Wunsch wider, eben diese Werte zu maximieren. Als Fitnessfunktion wird nun eine gewichtete Summe aus diesen beiden Werten verwendet. Dabei wird die Sensitivität etwas stärker berücksichtigt als die Spezifität. Dies erscheint notwendig, da es wichtig ist, CpG-Inseln aufzufinden, auch wenn die Spezifität darunter etwas leidet. Dies verringert die Wahrscheinlichkeit CpG-Inseln nicht zu entdecken. Hier wird eine Gewichtung von 60% zu 40% zugunsten der Sensitivität gewählt. Damit ergibt sich die Fitnessfunktion wie folgt:

$$\Psi_3(\mathcal{I}_i) := 0{,}6 \cdot \text{Sensitivität} + 0{,}4 \cdot \text{Spezifität}, \tag{4.4}$$

wobei Sensitivität und Spezifität die Ergebnisse der Berechnung nach Gleichung 3.2 bzw. 3.3 bezeichnen.

4.6.1 Ermittlung günstiger Parameter mittels SPO

Wie bereits für die Fitnessfunktion aus Gleichung 4.2, soll nun auch für diese Fitnessfunktion eine günstige Einstellung der Parameter für den EA gefunden werden. Dazu wird wieder die sequentielle Parameteroptimierung genutzt. Die anzupassenden Parameter sind hier die gleichen wie schon in Kapitel 4.2, also die Populationsgröße sowie die Mutations- und Rekombinationswahrscheinlichkeit.

Es wird auch hier ein anisotropisches Modell verwendet und eine Iteration der SPO durchgeführt. In diesem Fall werden neun Designpunkte mit jeweils zwei Wiederholungen mit unterschiedlichen Seeds gewählt und für jede Parametereinstellung soll diesmal eine von der Populationsgröße abhängige Anzahl von Generationen berechnet werden. Dies bietet — wie bereits in Kapitel 4.3 erläutert — den Vorteil für jeden Designpunkt eine ähnliche Rechenzeit zur Verfügung zu stellen, sodass überprüft wird, welche Parametereinstellung in einer bestimmten Zeit die besten Resultate aufweist. Dies wird realisiert, indem eine feste Anzahl von Individuen vorgegeben wird, für die die Fitnesswerte berechnet werden sollen. Wird diese durch die Anzahl der Individuen pro Generation geteilt, ergibt sich — nach oben aufgerundet — die Anzahl Generationen, die erzeugt werden sollen. Hier werden dazu 80 Individuen gewählt, so dass beispielsweise bei einer Populationsgröße von zwei demnach 40 Generationen berechnet werden, während es bei einer Populationsgröße von zehn lediglich acht Generationen sind.

Die Ergebnisse aus der SPO sind in Abbildung 4.10 und 4.11 aufgetragen. Besonders auffällig ist, dass die Fitnesswerte hauptsächlich im Bereich von 60 bis 62 liegen. Bei genauerer Betrachtung der Fitnessfunktion wird deutlich, dass ein Fitnesswert von 60 bereits durch das Maximieren der Sensitivität erreicht wird. Praktisch würde es dazu ausreichen, alle Nukleotide als zu CpG-Inseln gehörend zu markieren. Auf diese Art

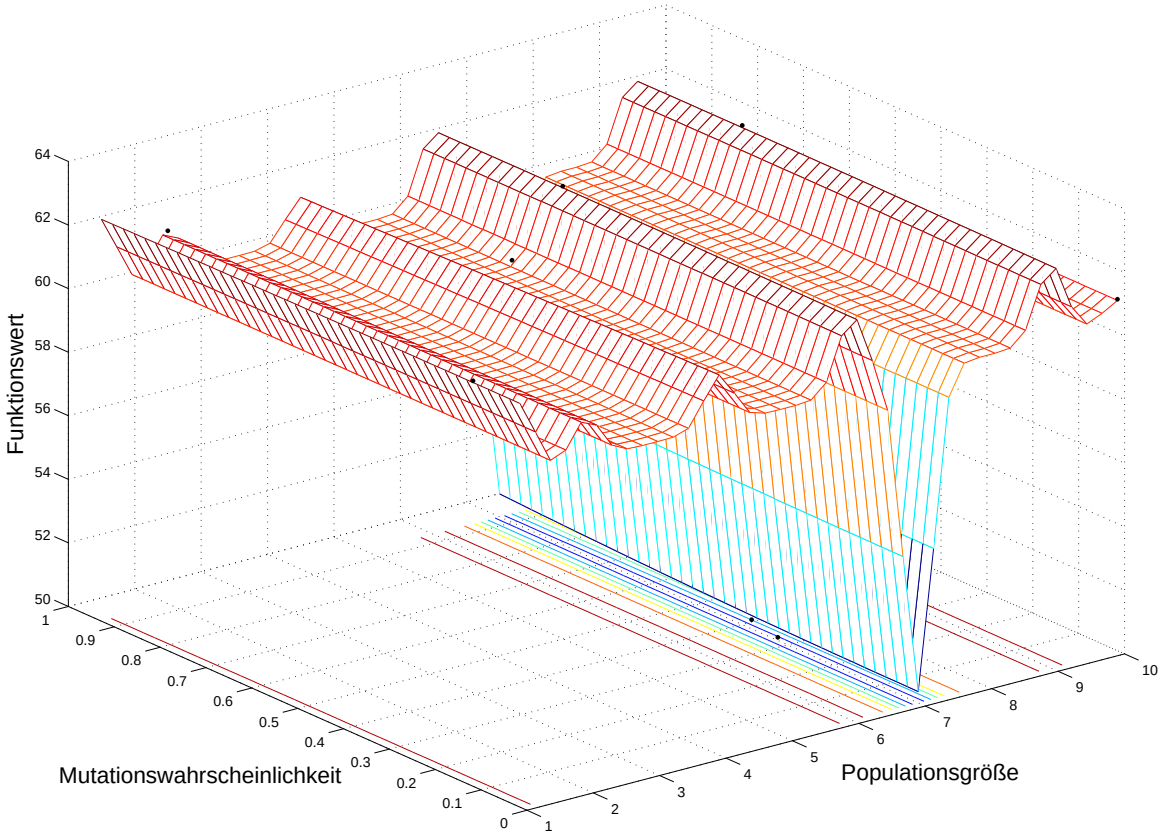


Abbildung 4.10: *Funktionswert Y in Abhängigkeit von der Populationsgröße und der Mutationswahrscheinlichkeit bei Fitnessfunktion* Ψ_3. Die Werte basieren auf einem Design mit neun Designpunkten für die jeweils zwei Läufe mit unterschiedlichen Random-Seeds berechnet werden. Die Anzahl durchgeführter Generationen ist von der Populationsgröße abhängig, sodass für jeden Designpunkt für etwa 80 Individuen die Fitness bestimmt wird. Die Mutationswahrscheinlichkeit wirkt sich nicht auf den Funktionswert aus.

würden alle tatsächlichen CpG-Inseln von der Vorhersage abgedeckt werden und die Sensitivität läge bei 100%. Diese Vermutung soll später durch Ermittlung der einzelnen Werte für Sensitivität und Spezifität sowie der Anteile von falschen bzw. verpassten CpG-Inseln überprüft werden.

Außerdem sticht in den beiden Abbildungen der fehlende Einfluss von Mutations- und Rekombinationswahrscheinlichkeit auf den Funktionswert heraus. In Abbildung 4.11 zeigt sich der Funktionswert völlig unabhängig von diesen beiden Parametern. Die Mutationswahrscheinlichkeit hat jedoch im Allgemeinen Einfluss auf die erzeugten Individuen und die Schritte im Suchraum. Die Populationsgröße weist ebenfalls nur sehr geringe Auswirkungen auf den Funktionswert auf. Der einzige Ausreißer bei einer Populationsgröße von sieben könnte zufällig bedingt sein und bei einer größeren Anzahl

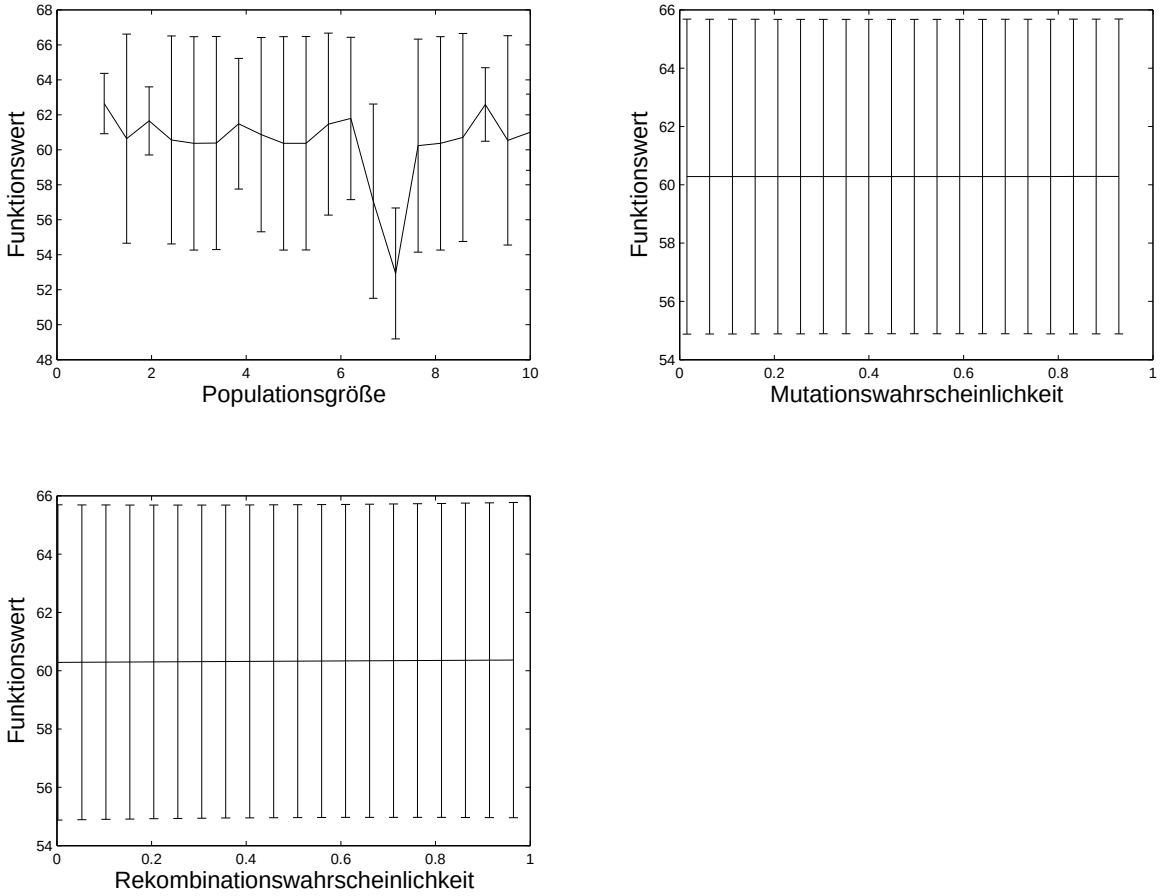


Abbildung 4.11: *Funktionswert Y in Abhängigkeit von den angepassten Parametern Populationsgröße, Mutations- und Rekombinationswahrscheinlichkeit bei Fitnessfunktion* Ψ_3. Auch hier liegt das Design mit neun Designpunkten zugrunde. Für jeden Designpunkt werden in zwei unabhängigen Läufen für etwa 80 Individuen die Fitness bestimmt. Mutations- und Rekombinationswahrscheinlichkeit zeigen keinen Einfluss auf den Fitnesswert.

von Läufen mit verschiedenen Random-Seeds verschwinden. Die beobachtete weitestgehende Unabhängigkeit des Funktionswertes von den Parametern würde sich allerdings ebenfalls durch die oben schon erwähnte Vermutung erklären lassen.

4.6.2 Überprüfung der Güte der HMMs

Für die Überprüfung der Ergebnisse, die mit dieser Fitnessfunktion erzielt wurden, wird aus den neun verschiedenen Parametereinstellungen an den einzelnen Designpunkten der SPO jeweils das am höchsten bewertete HMM für die weitere Betrachtung herangezogen. Für diese werden dann wiederum die in Abschnitt 3.4 beschriebenen Gütemaße bestimmt, ebenfalls einmal unter Verwendung der Viterbi-Dekodierung und einmal mit Krogh-Dekodierung. Diese sind in Tabelle 4.3 und 4.4 zusammengefasst.

Sensitivität in %	Spezifität in %	verpasste CpGs in %	falsche CpGs in %
100	8,55	0	74,36
99,50	8,54	0	86,43
100	8,55	0	74,19
99,80	8,57	0	92,93
100	8,55	0	74,19
100	8,55	0	74,36
97,01	8,49	0	88,24
100	8,55	0	75,31
100	8,55	0	74,19

Tabelle 4.3: *Gütemaße für Viterbi-Dekodierung für die jeweils besten Ergebnisse der neun Designpunkte der SPO.* Für jeden Designpunkt wird das höchstbewertete Individuum der letzten berechneten Generation zur Berechnung der Gütemaße verwendet. Für keinen Designpunkt wird eine CpG-Insel verpasst.

Bei der Viterbi-Dekodierung beträgt die Sensitivität in sechs der neun Fälle 100%, bei den restlichen HMMs lag sie jedoch auch über 97%. Dies bestätigt die Vermutung, dass lediglich die Sensitivität maximiert wurde. Die Spezifität liegt zwischen 8,49% und 8,57%. Unter Berücksichtigung der Tatsache, dass in den Validierungsdaten nur etwa 7% der Nukleotide in CpG-Inseln liegen, wird schnell klar, dass tatsächlich fast alle Nukleotide als in CpG-Inseln liegend markiert wurden. Für die Krogh-Dekodierung ergibt sich ein sehr ähnliches Bild.

Für beide Dekodierungsvarianten werden in nahezu allen Fällen alle tatsächlichen CpG-Inseln von einer vorhergesagten überlappt, auch dies entspricht der Vermutung. Der Anteil von falschen CpG-Inseln ist dabei immer sehr groß. In allen getesteten HMMs liegt er über 74,36%. Es zeigt sich also, dass diese Fitnessfunktion ebenfalls noch nicht den Anforderungen entspricht. Daher wird die Idee der Miteinbeziehung der Gütemaße aufgrund ihrer unzweckmäßigen Resultate wieder verworfen.

Sensitivität in %	Spezifität in %	verpasste CpGs in %	falsche CpGs in %
100	8,55	0	74,19
100	8,55	0	74,19
0	0	100	100
100	8,55	0	74,19
100	8,55	0	74,19
100	8,55	0	74,19
100	8,55	0	74,19
100	8,55	0	74,52
100	8,55	0	74,19

Tabelle 4.4: *Gütemaße für Krogh-Dekodierung für das jeweils beste Ergebnis der neun Designpunkte.* Für jeden Designpunkt wird das höchstbewertete Individuum der jeweils letzten berechneten Generation für die Ermittlung der Gütemaße herangezogen.

4.7 Vierte Fitnessfunktion

Bei den bisher betrachteten Fitnessfunktionen fiel auf, dass sich zumeist Extremsituationen einstellten. Entweder wurden alle oder gar keine Nukleotide als in CpG-Inseln liegend markiert. Aufgrund dieser Problematik, erscheint es wichtig, die aus der zweiten Fitnessfunktion bekannte Bewertung mit Gewichten zu versehen. Auf diese Weise sollen sowohl das Auffinden von CpG-Inseln, als auch die Identifizierung von Nicht-CpG-Inseln möglichst gleich stark auf die Fitnessfunktion einwirken.

Um dies zu realisieren, wird ähnlich vorgegangen wie bei der zweiten Fitnessfunktion, allerdings wird jetzt nicht mehr jede Übereinstimmung von Vorhersage und tatsächlicher Markierung, unabhängig davon, ob es sich in beiden Fällen um eine 1 für eine CpG-Insel oder sonst um eine 0 handelt, gleich stark bewertet. Stattdessen wird die folgende Gewichtung gewählt.

Ein korrekt als in einer CpG-Inseln liegend vorhergesagtes Nukleotid soll die höchste Belohnung bekommen, da dies den erwünschten Fall darstellt. In diesem Fall wird dafür die Belohnung 1 vergeben. Da sich jedoch bei den vorliegenden Sequenzen der Großteil der Nukleotide außerhalb von CpG-Inseln befindet, muss die Belohnung für ein nicht in einer CpG-Insel liegendes Nukleotid, das ebenfalls richtig als solches vorhergesagt wird, weitaus geringer ausfallen, um nicht wieder in den oben beschriebenen Extremfall zu gelangen. Da nur ca. 6,9% der Nukleotide in CpG-Inseln liegen, sollte der Unterschied in der Belohnung in etwa dieses Verhältnis repräsentieren, d. h. die Belohnung wird hier als $1 \cdot \frac{6,9}{93,1} \approx 0,074$ gewählt.

Die beiden Fälle, in denen falsch klassifiziert wird, sind zwar beide nicht erwünscht, dennoch soll auch hier eine Unterscheidung vorgenommen werden. Da die CpG-Inseln

gesucht werden, sollen diese besser etwas großzügiger vorhergesagt werden, als dass sie übersehen werden. Ein Nukleotid innerhalb einer CpG-Insel, das falsch vorhergesagt wird, soll gar keine Belohnung erhalten; wird hingegen für ein Nukleotid außerhalb einer CpG-Insel eine falsche Vorhersage getroffen, soll noch eine minimale Belohnung vergeben werden, hier 0,001, um diesen Fall dem vorherigen vorzuziehen.

Die auf diese Weise für jedes Nukleotid entlang einer Beispielsequenz x^j vergebenen Belohnungen werden addiert. Anschließend wird die Summe mit der Länge l^j dieser Sequenz skaliert, um jeder Sequenz den gleichen Einfluss auf den Fitnesswert zu ermöglichen und mit 100 multipliziert. Zum Schluss wird noch der Durchschnitt über alle k Trainingsbeispiele gebildet. Es ergibt sich die Fitnessfunktion

$$\begin{aligned}\Psi_4(\mathcal{I}_i) := \frac{1}{k}\sum_{j=1}^{k}\frac{100}{l^j}\cdot\Big[&1{,}000\cdot\|\{x_i^j|1\le i\le l^j\wedge y_i^j=1\wedge label_{viterbi}(x_i^j)=1\}\|\\ &+0{,}074\cdot\|\{x_i^j|1\le i\le l^j\wedge y_i^j=0\wedge label_{viterbi}(x_i^j)=0\}\|\\ &+0{,}001\cdot\|\{x_i^j|1\le i\le l^j\wedge y_i^j=0\wedge label_{viterbi}(x_i^j)=1\}\|\\ &+0{,}000\cdot\|\{x_i^j|1\le i\le l^j\wedge y_i^j=1\wedge label_{viterbi}(x_i^j)=0\}\|\Big].\end{aligned}\quad(4.5)$$

4.7.1 Anpassung der Parameter mithilfe von SPO

Auch für die Fitnessfunktion aus Gleichung 4.5 wird nun die SPO angewendet, um möglichst günstige Werte für die Parameter Populationsgröße, Mutations- und Rekombinationswahrscheinlichkeit zu finden. Dazu werden für die SPO die gleichen Einstellungen verwendet wie in Abschnitt 4.6.1. Unter Verwendung eines anisotropischen Modells wird eine Iteration der SPO durchgeführt. Dazu werden an zehn Designpunkten jeweils zwei verschiedene Seeds gewählt und jeweils — wie bereits zuvor — so viele Generationen berechnet, dass etwa 80 Individuen erzeugt und evaluiert werden. Die Populationsgröße wird wieder zwischen zwei und zehn variiert, während für Mutations- und Rekombinationswahrscheinlichkeit Wertebereiche zwischen 0 und 1 gewählt werden.

In Abbildung 4.12 ist der Funktionswert in Abhängigkeit von der Populationsgröße und der Mutationswahrscheinlichkeit aufgetragen. Die Funktionswerte liegen hier alle in einem sehr kleinen Wertebereich. Das Maximum liegt bei einer Populationsgröße von 6 oder 7 und einer Mutationswahrscheinlichkeit um 0,3.

Abbildung 4.13 zeigt noch einmal den Einfluss jedes einzelnen veränderlichen Parameters auf den Funktionswert. Im Vergleich zu der zuvor verwendeten Fitnessfunktion Ψ_3 fällt auf, dass die Mutations- und Rekombinationswahrscheinlichkeit nun einen deutlich größeren Einfluss haben. Die Diversität, die sich aus den großen Populationsgrößen ergibt, ist für eine relativ hohe Rekombinationswahrscheinleichkeit vorteilhaft. Die sich daraus ergebenden Individuen erzielen eine gute Bewertung mit der Fitnessfunktion.

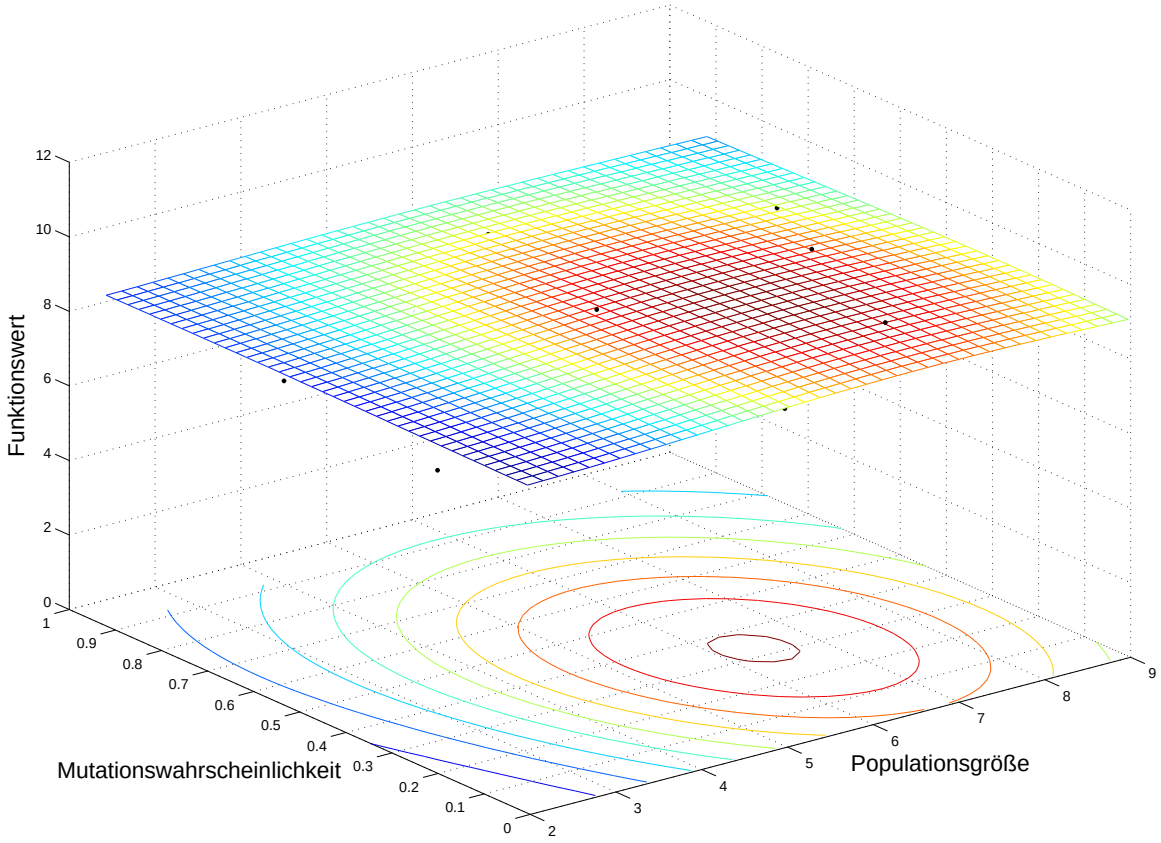


Abbildung 4.12: *Funktionswert in Abhängigkeit von Populationsgröße und Mutationswahrscheinlichkeit bei der Fitnessfunktion* Ψ_4. Die Werte basieren auf einem Design mit zehn Designpunkten. Für jeden dieser Designpunkte werden in zwei voneinander unabhängigen Läufen so viele Generationen berechnet, dass für etwa 80 Individuen die Fitness bestimmt wird.

Dies spiegelt sich in dem Maximum des Fitnesswertes bei einer Rekombinationswahrscheinlichkeit von etwa 0,75 wider. Der Abfall des maximalen Fitnesswertes bei Rekombinationswahrscheinlichkeiten größer als 0,8 lässt darauf schließen, dass auch Mutationen ohne anschließende Rebkombination von essentieller Bedeutung sind.

Insgesamt entspricht dieser Einfluss der verschiedenen Parameter auf den Funktionswert dem, was von einer adäquaten Fitnessfunktion erwartet wird. Für die folgenden Untersuchungen zur Güte der Lösungen wird eine Populationsgröße von sieben gewählt, da dort der maximale Funktionswert auftritt. Die Mutationswahrscheinlichkeit wird auf 0,25 und die Rekombinationswahrscheinlichkeit auf 0,75 gesetzt, da sich dies im Hinblick auf die Ergebnisse der SPO als günstig erwiesen hat.

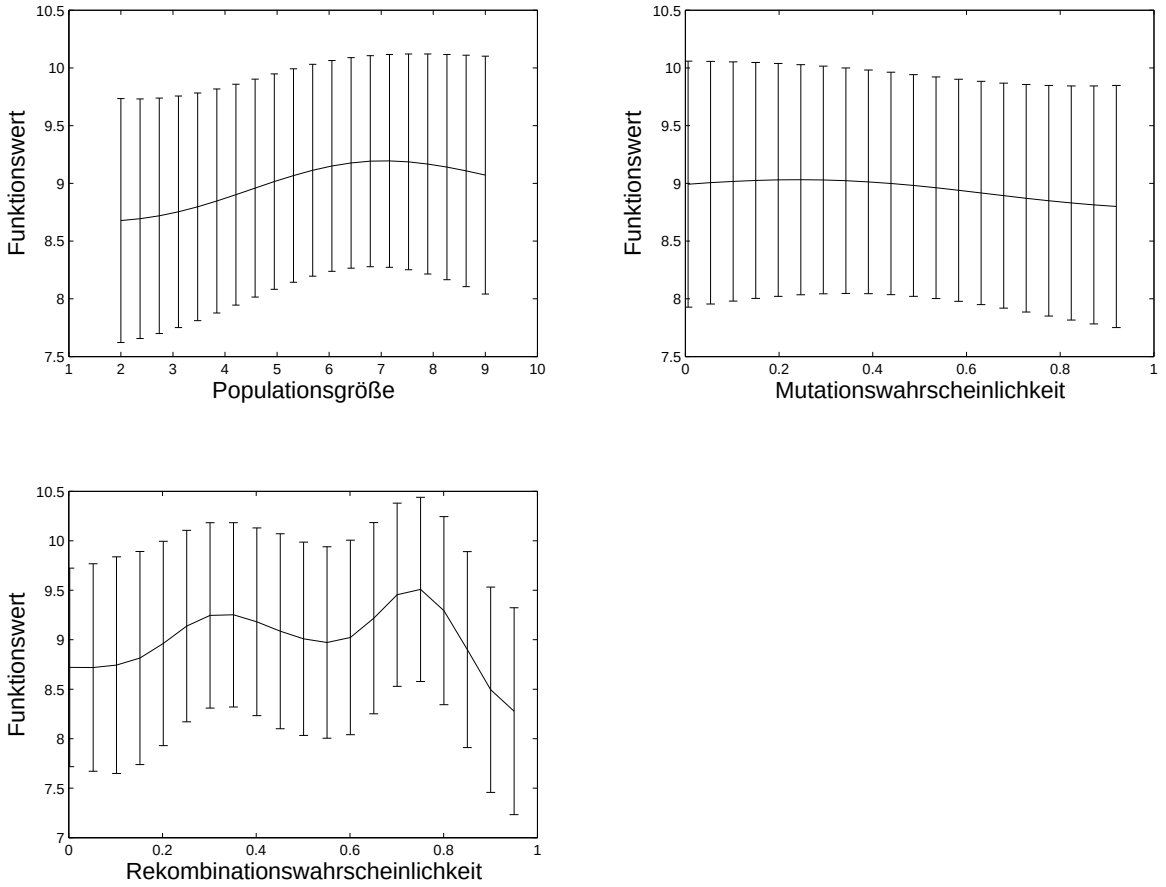


Abbildung 4.13: *Funktionswert in Abhängigkeit von den drei angepassten Variablen bei Fitnessfunktion* Ψ_4. Hier liegt ebenfalls das Design mit zehn Designpunkten, für die jeweils zwei unabhängige Läufe mit jeweils ca. 80 Individuen, für die die Fitness berechnet wird, gestartet werden.

4.7.2 Güte der erzeugten Lösungen

Um die Güte der durch diesen Ansatz erzeugten Lösungen abschätzen zu können, werden nun für die ermittelten Parameterwerte drei unabhängige Läufe mit verschiedenen Random-Seeds gestartet. Es werden dabei jeweils 100 Generationen berechnet. Für jeden Lauf werden in jeder zehnten Generation für das Individuum mit der aktuell höchsten Fitness die Gütemaße aus Abschnitt 3.4 bestimmt und über die Läufe gemittelt. Diese Werte sind in Tabelle 4.14 aufgetragen.

Die Sensitivitätswerte sind mit 70% bis 90% deutlich höher als bei der zweiten Fitnessfunktion, bei der die verschiedenen Arten von Übereinstimmungen und Fehlern gleich stark in die Fitness eingingen. Andererseits erreichen sie nicht ganz die bei der dritten Fitnessfunktion erreichte Güte. Dort bildeten jedoch ausschließlich diese Werte die Fitnessfunktion. Die Spezifität ist über die Generationen relativ konstant, mit ei-

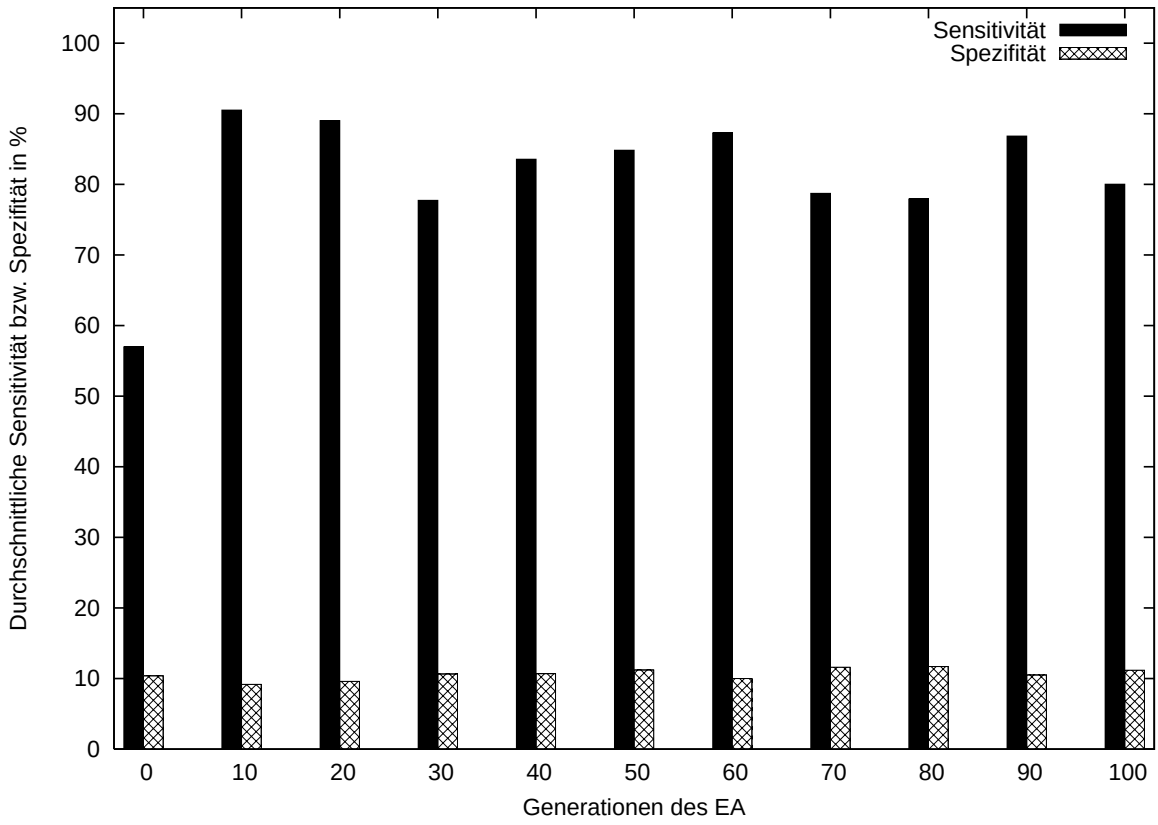


Abbildung 4.14: *Sensitivität und Spezifität der Ergebnisse bei der Fitnessfunktion* Ψ_4. Es werden drei unabhängige Läufe mit jeweils 100 Generationen gestartet. Es werden die Parameter verwendet, die sich in der SPO als besonders vorteilhaft erwiesen haben: Die Populationsgröße wird auf sieben, die Mutationswahrscheinlichkeit auf 0,25 und die Rekombinationswahrscheinlichkeit auf 0,75 gesetzt. Nach jeder zehnten Generation werden für die jeweils höchstbewerteten Individuen der drei Läufe die Sensitivität und die Spezifität bestimmt und über die Läufe gemittelt. Diese Mittelwerte sind hier aufgetragen.

nem Wert von ca. 10 jedoch auf einem recht niedrigen Niveau, wenn auch etwas höher als bei der dritten Fitnessfunktion. Das fehlende Ansteigen der Spezifität könnte in der Wahl des Selektionsoperators begründet liegen. Dieser verwirft teilweise die besten Individuen einer Generation, wie auch an den Schwankungen der Fitnessfunktion in Abbildung 4.16 auf Seite 66 zu sehen ist. Hierzu wird im späteren Verlauf noch ein zweiter Selektionsoperator getestet.

Generation	verpasste CpGs in %	falsche CpGs in %
0	0	91,86
10	0	86,56
20	0	88,82
30	0	92,01
40	2,11	86,06
50	2,53	83,45
60	2,53	82,05
70	2,53	81,90
80	2,53	82,24
90	2,11	82,27
100	2,53	81,51

Tabelle 4.5: *Verpasste und falsche CpG-Inseln der Ergebnisse bei der Fitnessfunktion* Ψ_4. Nach jeder zehnten Generation werden mithilfe der Viterbi-Dekodierung für jeden der drei unabhängigen Läufe die Gütemaße bestimmt und über die Läufe gemittelt.

In Tabelle 4.5 sind die mittleren Werte für die verpassten und falschen CpG-Inseln nach jeder zehnten Generation dargestellt. Die Tabelle zeigt, dass hier von Anfang an fast alle realen CpG-Inseln auch von einer vorhergesagten überlappt werden. Es werden maximal 2,53% der tatsächlichen CpG-Inseln nicht von einer vorhergesagten überlappt. Gleichzeitig ist der Anteil von falschen CpG-Inseln mit 80% bis 90% sehr groß. Dies bedeutete, dass insgesamt sehr viele CpG-Inseln vorhergesagt werden. Bei näherer Betrachtung der vorhergesagten Sequenzen erklärt sich diese hohe Anzahl an CpG-Inseln: Es werden sehr viele CpG-Inseln mit nur ein paar wenigen Nukleotiden vorhergesagt. In der Realität vorkommende Inseln weisen meist eine Länge von mehreren hundert Nukleotiden und mehr auf. Die hohe Anzahl dieser kurzen fälschlicherweise vorhergesagten CpG-Inseln könnte die Hauptursache für die geringe Spezifitiät darstellen. Im Folgenden wird zunächst dieses Problem untersucht.

4.7.3 Nachbearbeitung der Ergebnisse

Bei der Betrachtung der mittels Viterbi-Algorithmus für die Ergebnis-HMMs des EA erzeugten Vorhersagen finden sich sehr viele CpG-Inseln, die nur einige wenige Nu-

kleotide umfassen. Dies entspricht jedoch keineswegs der Realität. Abhilfe soll nun ein Nachbearbeitungsschritt schaffen. Dieser wird im Anschluss an die Berechnung der Markierungen mit dem Viterbi-Algorithmus ausgeführt. Er soll einerseits besonders kleine CpG-Inseln entfernen und andererseits kleine Lücken in größeren Inseln schließen.

Für eine CpG-Insel erscheinen folgende drei Eigenschaften sinnvoll. Erstens beginnt und endet sie in einem Nukleotid, das mit der Markierung 1, also zu einer CpG-Insel gehörend, versehen wurde. Zweitens bilden Bereiche zwischen zwei solchen Nukleotiden nur dann eine CpG-Insel, wenn mindestens ein bestimmter prozentualer Anteil der Nukleotide mit einer 1 markiert wurde. Der genaue Anteil kann dabei variiert werden. Es sollten auf jeden Fall mehr als 50% sein, da dann mehrheitlich die Markierung 1 vorkommt. Wird allerdings 100% gewählt, werden keine Lücken mehr geschlossen. Eine Erhöhung des Anteils führt sowohl zu einem Ansteigen der Spezifität, als auch zu einem gleichzeitigen Abfall der Sensitivität. Es zeigte sich, dass ein Anteil von 70% bis 80% zu guten Ergebnissen führt.

Drittens sollte die Länge einer CpG-Insel einen bestimmten Schwellenwert überschreiten. Hierbei ist es ebenfalls wichtig, einen ausgewogenen Wert zu finden: Wird der Schwellenwert zu niedrig gewählt, gibt es weiterhin sehr viele kleine CpG-Inseln. Ein zu hoher Schwellenwert dagegen bewirkt, dass nur noch wenige CpG-Inseln gefunden und unter Umständen tatsächliche CpG-Inseln verpasst werden. Hier hat sich ein Schwellenwert von 100 als vorteilhaft erwiesen.

Nur Bereiche, die diese drei Anforderungen erfüllen, werden als CpG-Inseln markiert. Die in ihnen liegenden Nukleotide werden mit der Markierung 1 versehen, während alle anderen Nukleotide mit 0 markiert werden. Im Folgenden werden für einen Anteil von 70% Nukleotiden mit Markierung 1 in CpG-Inseln ein Schwellenwert von 70% und bei der Länge ein Schwellenwert von 100 Nukleotiden gewählt.

Um diese Bereiche zu finden werden die vom Viterbi-Algorithmus ausgegebenen Markierungen sequentiell abgearbeitet. Beginnend bei der ersten 1-Markierung wird ein Block gesucht. Dieser endet an der ersten Position, an der der Anteil von 1-Markierungen in dem Block den vorgegebenen Schwellenwert unterschreitet. Sollte dieses Blockende mit einer 0-Markierung zusammenfallen, wird das Blockende auf die letzte vorkommende 1-Markierung zurückgesetzt. Beträgt die Länge des Blocks mindestens den vorgegebenen Schwellenwert, so wird der gesamte Block als CpG-Insel markiert, andernfalls wird der komplette Block mit 0 markiert. Dieses Verfahren wird iterativ auf die restliche Sequenz hinter dem Block angewendet.

Die Auswirkungen hiervon auf die Güte der Lösungen werden wieder mit Hilfe der Gütemaße aus Abschnitt 3.4 untersucht. Dazu wird für die drei unabhängigen Läufe nach jeder zehnten Generation auf das höchstbewertete Individuum mithilfe der Viterbi-

Dekodierung vorläufige Markierungssequenzen erzeugt, auf diese jeweils der Nachbearbeitungsschritt angewendet und aufgrund der so bestimmten, endgültigen Markierungen die Gütemaße berechnet. Diese sind in Abbildung 4.15 zusammen mit den Ergebnissen ohne Nachbearbeitung aufgetragen, um einen direkten Vergleich zu erleichtern.

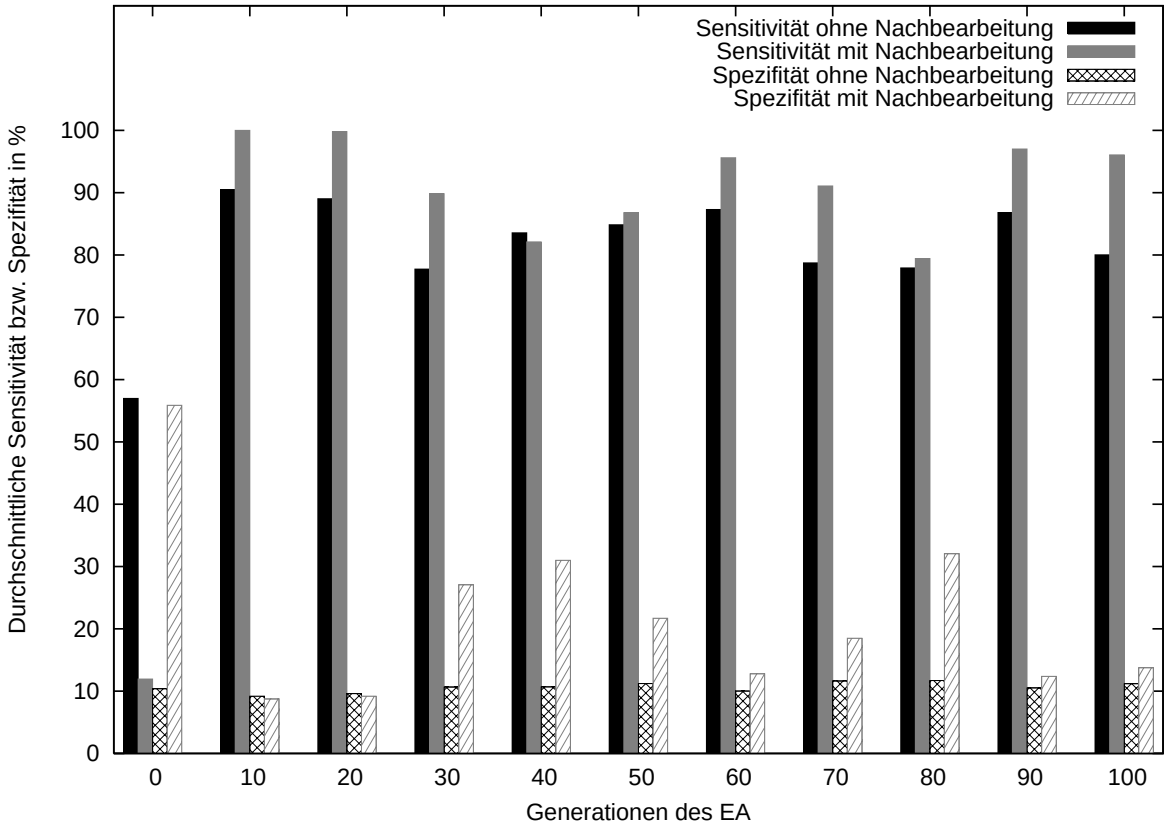


Abbildung 4.15: *Sensitivität und Spezifität ohne und mit Nachbearbeitung im Vergleich.* Die Werte ohne Nachbearbeitung entsprechen denen aus Abbildung 4.14. Für jede zehnte Generation der dort berechneten Läufe werden für das jeweils höchstbewerteten Individuum mithilfe der Viterbi-Dekodierung die vorläufige Markierungssequenzen bestimmt und anschließend auf diese der Nachbearbeitungsschritt angewendet. Für die resultierenden Vorhersagen werden die Gütemaße berechnet und über die Läufe gemittelt.

Die Sensitivität wird durch den Nachbearbeitungsschritt meist leicht erhöht oder bleibt annähernd gleich. Die Spezifität hingegen profitiert durch die Nachbearbeitung deutlich. Sie wird zum Teil mehr als verdreifacht. Hiermit steigt sie auf ähnliche Werte an, die auch durch den Baum-Welch-Algorithmus erreicht wurden, wobei die dort erreichte Sensitivität noch etwas höher ist als bei diesem Ansatz. Dabei ist jedoch zu berücksichtigen, dass die Berechnung einer Iteration des Baum-Welch-Algorithmus mehr als viermal so viel Rechenzeit benötigt, wie eine Generation des EAs mit den zuletzt verwendeten

Einstellungen. Damit kann dieser Ansatz im Vergleich zum Baum-Welch-Algorithmus als effektiv bezeichnet werden. Die Vermutung aus dem letzten Abschnitt, dass die geringe Spezifitiät in den vielen kleinen CpG-Inseln begründet liegt, hat sich demnach bewahrheitet. Daher erfolgt dieser Nachbearbeitungsschritt fortan immer, wenn nichts gegenteiliges erwähnt wird.

4.7.4 Variation der Populationsgröße

Um zu überprüfen, ob tatsächlich ein Einfluss der Populationsgröße auf die Fitnesswerte besteht und ob sich damit auch bei der Sensitivität und der Spezifität Auswirkungen bemerkbar machen, wird im Folgenden die Populationsgröße sieben, die sich bei der SPO als günstig erwiesen hat, mit einer Populationsgröße von zwei verglichen. Diese hat sich in der SPO als wenig vorteilhaft gezeigt.

Dazu werden — zusätzlich zu den bereits durchgeführten Läufen mit einer Populationsgröße von sieben — Läufe mit einer Populationsgröße von zwei durchgeführt. Dies geschieht wiederum für zwei verschiedene Random-Seeds, für die jeweils 175 Generationen berechnet werden. Die restlichen Parameter werden, genau wie im letzten Abschnitt bereits beschrieben, gewählt. Um die Ergebnisse von Läufen mit unterschiedlichen Populationsgrößen vergleichen zu können, ist hier wieder die Anzahl der Individuen ausschlaggebend, für die der Fitnesswert bestimmt wird. Die 175 Generationen mit zwei Individuen pro Generation entsprechen also 50 Generationen mit sieben Individuen pro Generation, da in beiden Fällen für 350 Individuen die Fitnesserte bestimmt werden. In Abbildung 4.16 ist für beide Populationsgrößen der Verlauf des maximalen Fitnesswertes — gemittelt über die Läufe mit unterschiedlichen Random-Seeds — in Abhängigkeit von der Anzahl der Individuen, für die die Fitness bestimmt wird, aufgetragen.

Es fällt auf, dass die Fitnesswerte, die bei der Populationsgröße zwei erreicht werden, durchweg unter den bei Populationsgröße sieben erzielten liegen. Eine mögliche Ursache hierfür ist, dass bei einer größeren Population mehr Individuen für die Erzeugung von Nachkommen zur Verfügung stehen. Hierbei könnte es von Vorteil sein, ein Individuum mit geringerer Fitness für die Mutation zu verwenden oder für die Rekombination zwei vermeintlich schlechtere Individuen zu verwenden. Diese könnten ihre positiven Eigenschaften weitergeben, während die schlechten Eigenschaften verloren gehen. Besteht die Population nur aus zwei Individuen, ist die Auswahl für die Erzeugung der Nachfolgegeneration deutlich geringer.

Außerdem zeigt Abbildung 4.16, dass es bei der kleineren Population zu deutlich größeren Schwankungen kommt. Dies wird unter Umständen durch den Selektionsoperator verursacht. Die Individuen für die Nachfolgegeneration werden zwar proportional zu ihrem Fitnesswert, jedoch trotzdem randomisiert ermittelt. Dadurch ist nicht gewährleistet, dass das Individuum mit dem höchsten Wert tatsächlich die Selektion überlebt. Aufgrund dessen ist es bei einer kleinen Populationsgröße und den damit verbundenen

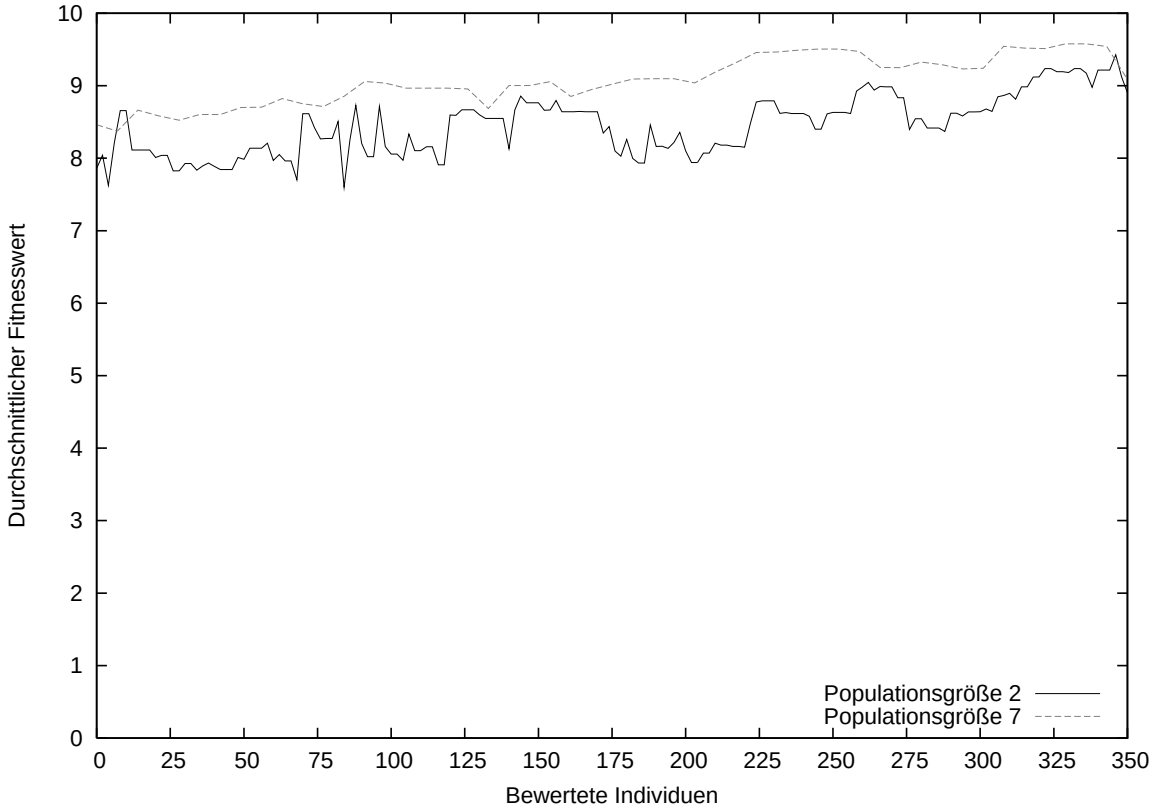


Abbildung 4.16: *Vergleich der Fitnesswerte für Populationsgrößen von zwei und sieben.* Die Fitnesswerte der unabhängigen Läufe mit Populationsgröße sieben, auf denen auch Abbildung 4.14 basiert, werden über die einzelnen Läufe gemittelt und aufgetragen. Zusätzlich werden noch zwei unabhängige Läufe mit Populationsgröße zwei und jeweils 175 Generationen durchgeführt. Die Mutations- und Rekombinationswahrscheinlichkeit werden nicht verändert. Die Fitnesswerte für die zusätzlichen Läufe werden ebenfalls gemittelt aufgetragen.

häufigeren Selektionsschritten weniger wahrscheinlich, dass es die größere Anzahl von Selektionen übersteht. Dies führt zu leichten Schwankungen des maximalen Fitnesswertes im Verlauf der durchgeführten Generationen. Allerdings sind die Schwankungen hier bei der Populationsgröße zwei sehr groß. Bei beiden getesteten Populationsgrößen steigt der maximale Fitnesswert außerdem nur sehr langsam an, so dass es sich anbietet, einen anderen Selektionsoperator zu testen. Dieser sollte deterministisch die Individuen für die nächste Generation auswählen, die den höchsten Fitnesswert erlangt haben. So sollte es zu einem schnelleren Anstieg der Fitness kommen.

Um die bisherigen Vermutungen zu untermauern, sind in Abbildung 4.17 zusätzlich für beide Populationsgrößen die durchschnittlichen Sensitivitäts- und Spezifitätswerte jeweils nach der Ausführung des Nachbearbeitungsschritts aufgetragen. Um dabei

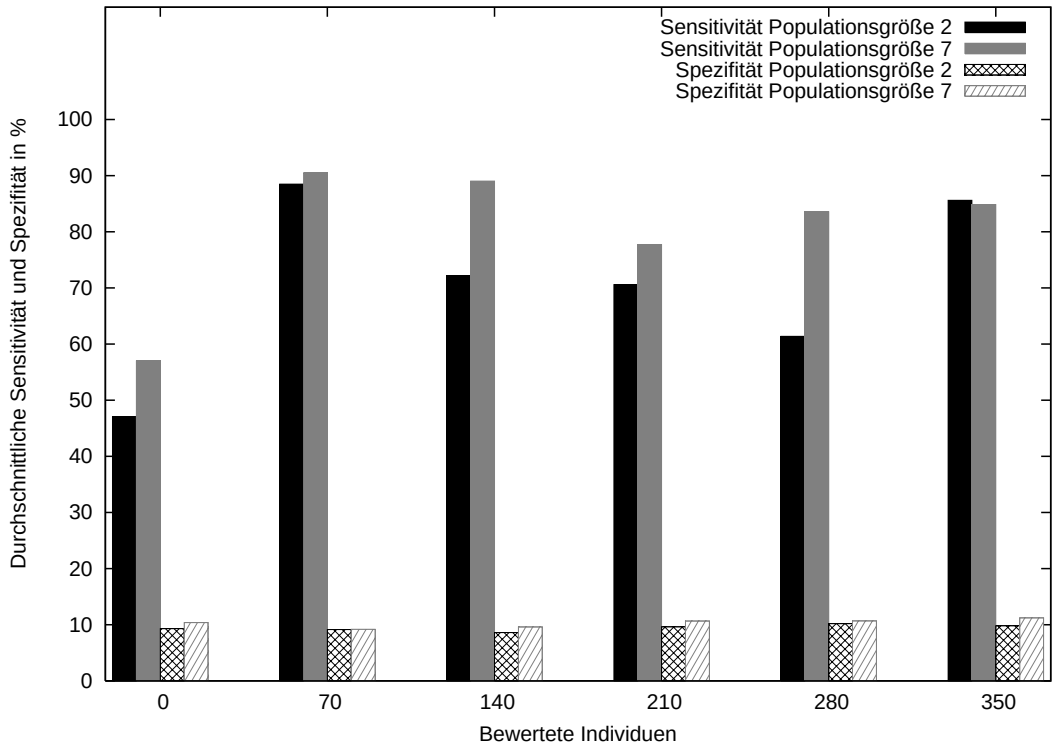


Abbildung 4.17: *Vergleich der Sensitivität und Spezifität bei Populationsgrößen von zwei und sieben.* Diese werden jeweils für die durchgeführten unabhängigen Läufe mit den unterschiedlichen Populationsgrößen basierend auf der Viterbi-Dekodierung mit Nachbearbeitungsschritt bestimmt und anschließend über die Läufe der jeweiligen Populationsgröße gemittelt.

einen Vergleich anstellen zu können, werden diese Werte nach jeweils 70 berechneten Fitnesswerten ermittelt, was bei zwei Individuen pro Generation 35 Generationen und bei sieben Individuen zehn Generationen bedeutet. Die Werte werden über die Läufe mit unterschiedlichen Random-Seeds gemittelt.

Auch hier zeigt sich die Überlegenheit der größeren Populationen. Für sie ist die Spezifität meist etwas höher als bei der Populationsgröße von zwei. Die Sensitivität ist zum Teil deutlich höher. Nach 350 Generationen sind die Gütemaße für beide Populationsgrößen nahezu gleich, was sich allerdings auch beim Fitnesswert bemerkbar machte. Der Fitnesswert für die Populationsgröße sieben sackte in der letzten berechneten Generation stark ab. Da der gewählte Selektionsoperator ein Absacken des maximalen Fitnesswertes ermöglicht, ist dieses Verhalten nicht ungewöhnlich. Insgesamt bestätigt sich allerdings das Ergebnis der SPO. Auch für die hier durchgeführten Läufe, für die jeweils anderen Random-Seeds als bei der SPO verwendet wurden, zeigt sich ein etwas besseres Verhalten für die größeren Populationen.

4.7.5 Deterministische Selektion

Die Überlegung, einen anderen Selektionsoperator zu verwenden, zeichnete sich bereits im letzten Abschnitt ab. Damit sollen die zum Teil großen Schwankungen der Fitness verhindert und möglicherweise ein schnellerer Anstieg der Fitnesswerte erreicht werden. Anstelle der in Abschnitt 4.1.4 beschriebenen modifizierten Rouletteradselektion, wird nun ein deterministischer Selektionsoperator verwendet. Dieser wählt in jeder Generation aus den n Individuen der Vorgängerpopulation und den n durch Mutation und Rekombination neu erzeugten Individuen die Individuen für die Nachfolgegeneration aus, die die n höchsten Fitnesswerte erhalten haben. In der Literatur wird dies auch als elitäre oder Abschneideselektion bezeichnet.

Dieser alternative Selektionsoperator stellt keinen Laufzeitzuwachs dar, da die Selektion der n besten Individuen schneller vonstatten geht als die fitnessproportionale Selektion mit Skalierung. Des Weiteren dominiert die Fitnessauswertung die Laufzeit, sodass der Einfluss des Selektionsoperators auf die Laufzeit ohnehin äußerst gering ausfällt.

Für die deterministische Selektion werden nun ebenfalls für zwei verschiedene Random-Seeds jeweils 100 Generationen berechnet. Dabei wird, wie in Abschnitt 4.7.2, eine Populationsgröße von sieben gewählt. Mutationswahrscheinlichkeit und Rekombinationswahrscheinlichkeit bleiben mit 0,25 und 0,75 ebenfalls gleich. In Abbildung 4.18 sind die gemittelten maximalen Fitnesswerte dieser Läufe nach jeder zehnten Generation noch einmal zusammen mit den gemittelten Fitnesswerten der Läufe aus Abschnitt 4.7.2 aufgetragen.

Bei der deterministischen Selektion steigt die Fitness deutlich schneller an. Schwankungen sind hier aufgrund ihrer Konstruktion nicht mehr möglich: Der maximale Fitnesswert der alten Population kann nur durch einen höheren Wert in der nächsten Generation abgelöst werden. Am Ende der durchgeführten 100 Generationen liegt der Wert für die deterministische Fitness bereits um etwa 25% höher als bei der modifizierten Rouletteradselektion.

Für die Ergebnisse der Läufe mit der deterministischen Selektion werden nun jeweils die Gütemaße gemäß Abschnitt 3.4 bestimmt und über die durchgeführten Läufe gemittelt. Die Werte für Sensitivität und Spezifität sind zusammen mit den Ergebnissen für die (randomisierte) modifizierte Rouletteradselektion noch einmal in Abbildung 4.19 aufgetragen. In beiden Fällen handelt es sich dabei um die Ergebnisse nach der Durchführung des Nachbearbeitungsschritts.

Hier zeigt sich, dass bei der deterministischen Selektion wesentlich geringere Schwankungen der Gütemaße auftreten. Bei der modifizierten Rouletteradselektion kommen öfter Ausreißer vor, bei denen die Spezifität deutlich absackt und die Sensitivität gleichzeitig ansteigt, während dies bei der deterministischen Selektion nicht mehr der Fall

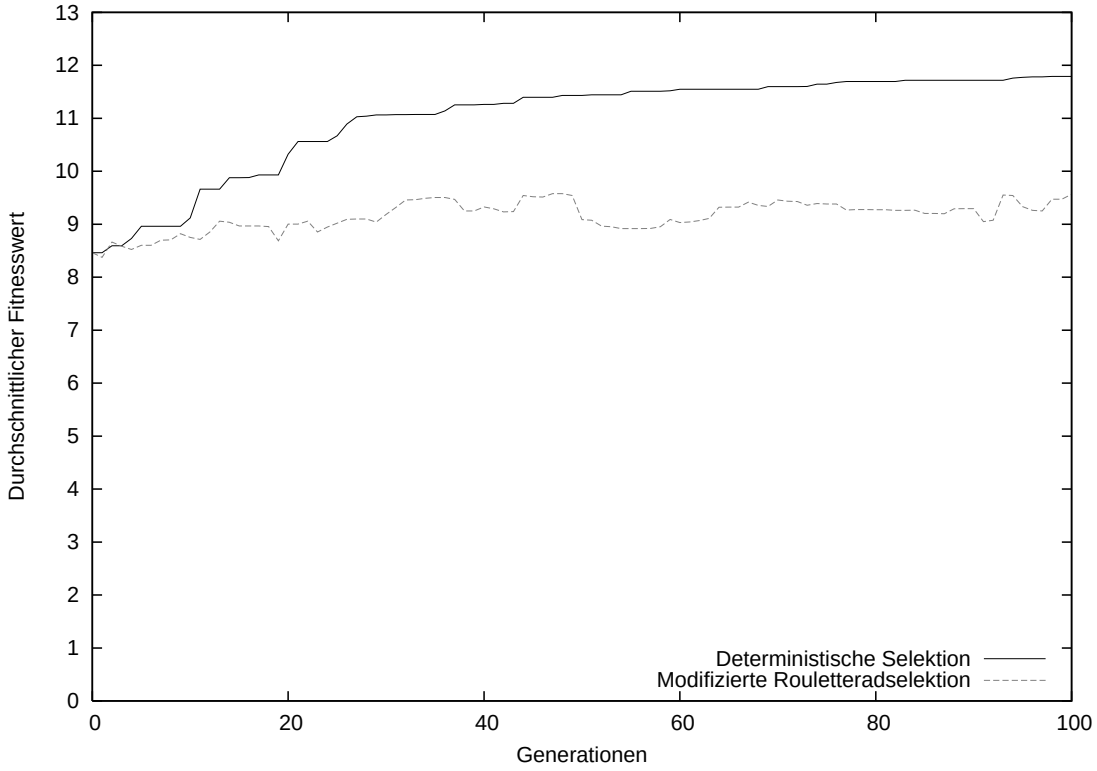


Abbildung 4.18: *Vergleich zwischen deterministischer und randomisierter Selektion.* Die Werte für die modifizierte Rouletteradselektion stimmen mit denen aus Abbildung 4.16 für die Populationsgröße sieben überein, für die die maximalen Fitnesswerte in der jeweiligen Generation über die unabhängigen Läufe gemittelt wurden. Zusätzlich wurden noch zwei Läufe mit unterschiedlichen Random-Seeds gestartet, bei denen die deterministische Selektion benutzt wird. Die übrigen Einstellungen werden beibehalten. Die Fitnesswerte unter Verwendung der deterministischen Selektion wurden ebenfalls über die Läufe gemittelt aufgetragen.

ist. Eine mögliche Erklärung dafür ist, dass bei der randomisierten Selektion auch die am schlechtesten bewerteten Individuen die Nachfolgegeneration ausmachen können. Bei der deterministischen Selektion kann dies nur ein Individuum schaffen, das besser bewertet ist als die Individuen aus der alten Generation. Insgesamt sind die Spezifitätswerte bei der deterministischen Selektion meist etwas höher, während bei ihr die Sensitivitätswerte etwas geringer ausfallen.

Die Mittelwerte für die verpassten und die falschen CpG-Inseln, die für das jeweils beste Individuum der jeweiligen Generation ermittelt wurden, sind in Tabelle 4.6 dargestellt. Hier fällt auf, dass der Anteil an tatsächlichen CpG-Inseln, die nicht von einer vorhergesagten CpG-Insel überlappt werden, sich auf einem relativ niedrigen Niveau einpendelt. Beim Baum-Welch-Algorithmus hingegen traten hier noch deutliche Schwankungen auf.

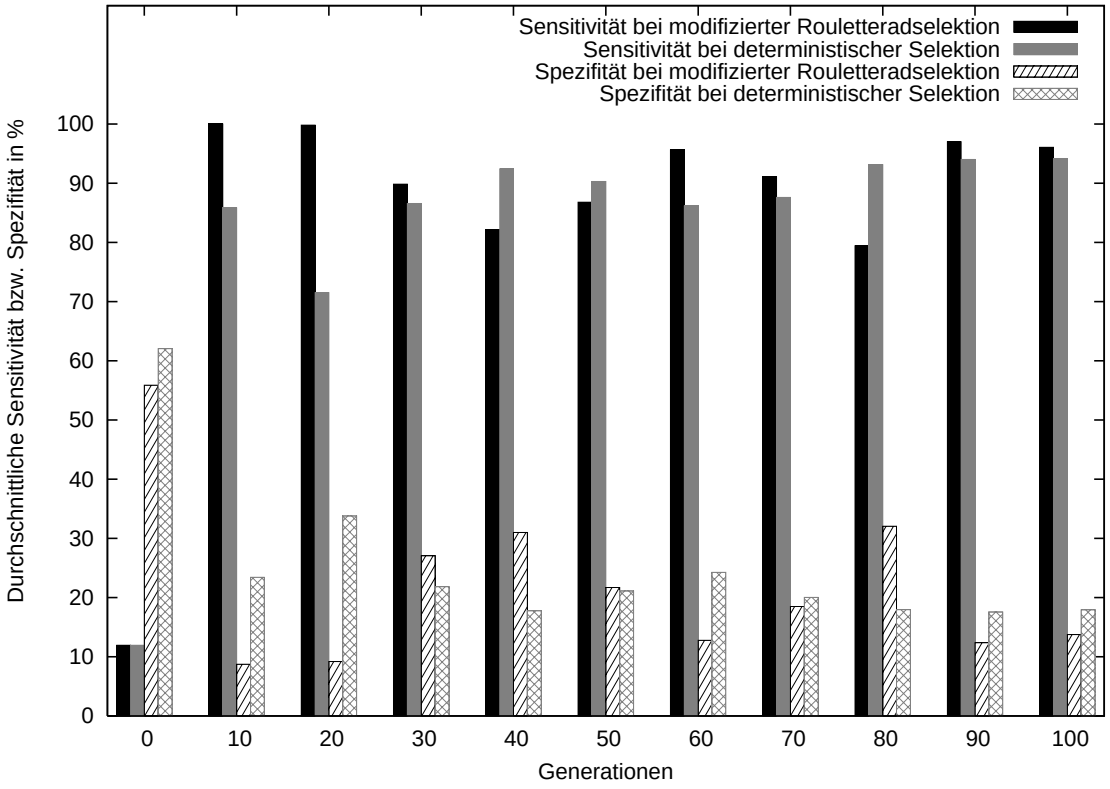


Abbildung 4.19: *Gütemaße für die modifizierte Rouletteradselektion und die deterministische Selektion jeweils nach dem Nachbearbeitungsschritt.* Für die unabhängigen Läufe mit den beiden Selektionsvarianten werden jeweils für das höchstbewertete Individuum jeder zehnten Generation mittels Viterbi-Algorithmus die vorläufigen Markierungssequenzen bestimmt und darauf der Nachbearbeitungsschritt angewendet. Basierend auf seinem Ergebnis werden die Gütemaße bestimmt. Die Werte für die beiden Selektionsoperatoren sind jeweils über die Läufe gemittelt aufgetragen.

Nach 80 Iterationen des Baum-Welch-Algorithmus lag der Anteil sogar noch über 56% (siehe Tabelle 3.1 auf Seite 33). Der Anteil an vorhergesagten CpG-Inseln, die nicht von einer tatsächlichen überlappt werden, ist bei dem Baum-Welch-Algorithmus und dem hier verwendeten EA in etwa gleich, obwohl er beim Baum-Welch-Algorithmus wiederum deutlich schwankte.

In Abbildung 4.20 und 4.21 werden exemplarisch für zwei der Sequenzen aus den Validierungsdaten die Ergebnisse der verschiedenen Methoden miteinander verglichen. Beide Abbildungen zeigen jeweils zunächst die bekannten tatsächlichen Markierungen, dann die Vorhersage unter Verwendung des mit 80 Iterationen des Baum-Welch-Algorithmus angepassten HMM und schließlich die Vorhersage unter Verwendung eines der Ergebnis-HMMs des zuletzt beschriebenen EAs nach 100 Generationen nach

Generationen	verpasste CpGs in %	falsche CpGs in %
0	79,74	35,21
10	6,75	75,84
20	19,83	72,94
30	7,17	79,06
40	4,22	75,19
50	5,91	71,62
60	7,17	70,25
70	6,75	70,43
80	2,53	70,70
90	2,11	71,56
100	3,38	70,90

Tabelle 4.6: *Verpasste und falsche CpG-Inseln für den EA mit deterministischer Selektion.* Die Werte werden jeweils für das höchstbewertete Individuum jeder Generation ermittelt, nachdem der Nachbearbeitungsschritt ausgeführt wurde und dann über die voneinander unabhängigen Läufe gemittelt.

der Ausführung des Nachbearbeitungsschritts. CpG-Inseln sind hier weiß dargestellt, während die übrigen Regionen der Sequenzen jeweils schwarz eingefärbt sind. Dies ermöglicht eine bessere Visualisierung als sie durch das Abdrucken der Markierungssequenz möglich wäre.

Abbildung 4.20: *Vergleich der Markierungen bei den verschiedenen Methoden für eine Sequenz der Validierungsdaten.* Die Originalsequenz ist dem Ergebnis des Baum-Welch-Algorithmus nach 80 Iterationen und einem Ergebnis-HMM des EAs mit deterministischer Selektion nach 100 Generationen gegenübergestellt. CpG-Inseln sind weiß dargestellt, während die übrigen Regionen schwarz eingefärbt sind.

Die Vorhersagen, die mit den beiden unterschiedlichen Methoden erzielt wurden, stimmen beide relativ gut mit der Originalsequenz überein. In Abbildung 4.20 findet sich in der Vorhersage des EAs eine kleinere CpG-Insel, die nicht von einer tatsächlichen überdeckt wird. Dies ist, wie sich auch schon in Tabelle 4.6 zeigte, in mehreren der Sequenzen aus den Validierungsdaten der Fall. Ansonsten gibt es sowohl Sequenzen,

Abbildung 4.21: *Vergleich der Markierungen bei den verschiedenen Methoden für eine zweite Sequenz der Validierungsdaten.* Oben: Die Originalsequenz aus den Validierungsdaten. Mitte: Ergebnis des Baum-Welch-Algorithmus nach 80 Iterationen. Unten: Ergebnis-HMM des EAs mit deterministischer Selektion nach 100 Generationen und anschließendem Nachbearbeitungsschritt. CpG-Inseln sind weiß dargestellt, während die übrigen Regionen schwarz eingefärbt sind.

bei denen die Vorhersage basierend auf dem Baum-Welch-Algorithmus besser mit dem Original übereinstimmt, als auch Sequenzen, bei denen die Vorhersage des EAs besser ist.

Insgesamt ergibt sich unter Verwendung des Baum-Welch-Algorithmus nach 80 Iterationen auf den Validierungsdaten eine Übereinstimmung der tatsächlichen mit den vorhergesagten Markierungen von 80,08%. Nach 100 Generationen des EAs ergibt sich eine Übereinstimmung von 71,39%. Hierbei sollte beachtet werden, dass die Laufzeit des Baum-Welch-Algorithmus deutlich höher ist. Nach etwa der gleichen Zeit, die für die Berechnung von 100 Generationen des zuletzt verwendeten EA benötigt wurden, lag die Übereinstimmung beim Baum-Welch-Algoritmus ebenfalls noch bei etwa 76%. Dies entspricht ca. 25 Iterationen.

4.7.6 Vergleich zwischen Viterbi- und Krogh-Dekodierung

Da nun eine Fitnessfunktion mit gutem Verhalten gefunden ist, soll nun für diese Überprüft werden, ob es für das Ergebnis dieses EAs einen Vorteil bietet, die Vorhersagen der Markierungen mithilfe der Krogh-Dekodierung zu treffen. Dieser Algorithmus stellt eine gute Approximation der Lösung der Gleichung 2.7 auf Seite 16 dar und dies entspricht dem eigentlichen Ziel bei der Vorhersage von CpG-Insel mittels HMMs.

Dazu werden für die jeweils höchstbewerteten Individuen der drei unabhängigen Läufe aus dem letzten Abschnitt die Gütemaße unter Verwendung des Krogh-Algorithmus berechnet und über die Läufe gemittelt. Dies geschieht nach jeweils zehn Generationen des EA, um auch einen Eindruck vom Verhalten der Gütemaße über die Zeit zu erlangen. Die Ergebnisse sind in Abbildung 4.22 zusammen mit den Ergebnissen aus dem letzten Abschnitt, für die die Viterbi-Dekodierung eingesetzt wurde, aufgetragen. Hierbei ist zu beachten, dass auf die durch den Krogh-Algorithmus bestimmte Markierungssequenz nicht mehr der Nachbearbeitungsschritt angewendet wird. Bei Betrach-

tung der resultierenden Markierungssequenzen und der ohnehin schon sehr geringen Sensitivität, wird klar, dass dieser hier keinen Vorteil bieten würde.

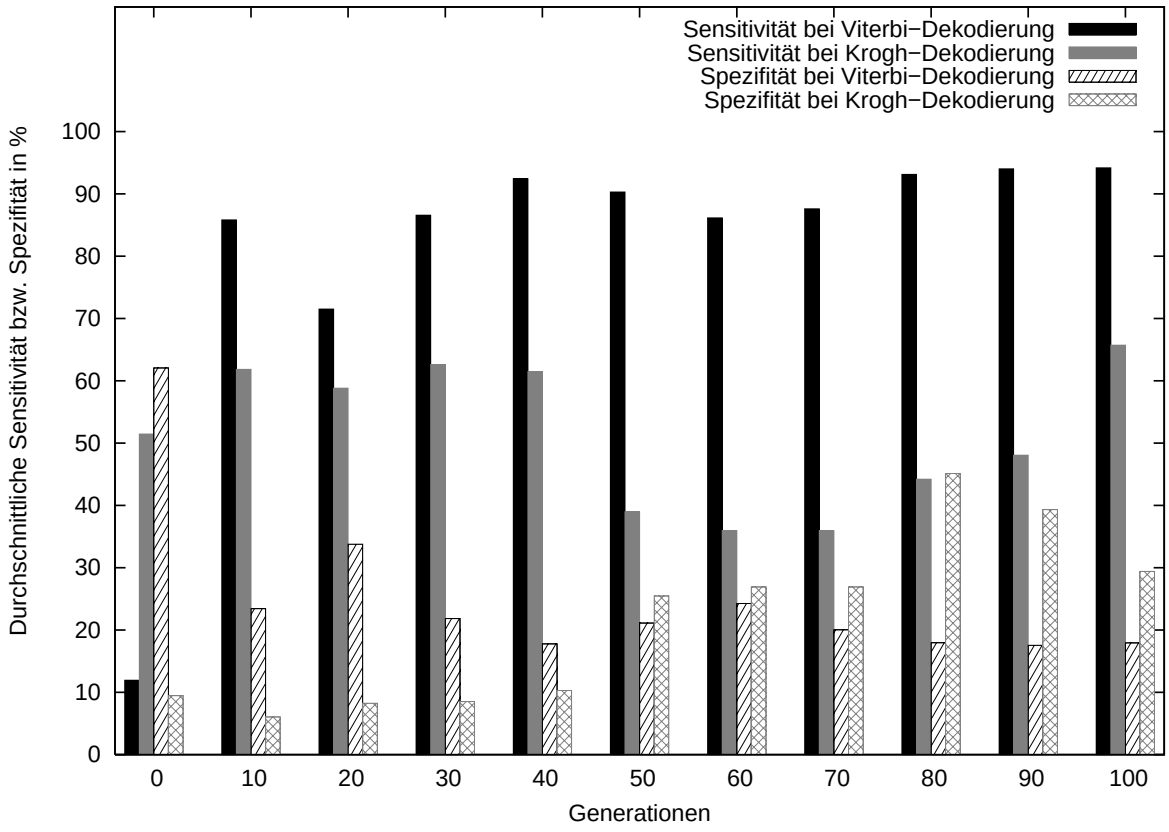


Abbildung 4.22: *Vergleich zwischen Viterbi- und Krogh-Dekodierung.* Für die zwei unabhängigen Läufe mit deterministischer Selektion, Populationsgröße sieben, Muationswahrscheinlichkeit 0,25 und Rekombinationswahrscheinlichkeit 0,75 werden wieder nach jeder zehnten Generation die Gütemaße für die jeweils höchstbewerteten Individuen bestimmt. Dazu wird einmal die Viterbi-Dekodierung mit anschließendem Nachbearbeitungsschritt und einmal die Krogh-Dekodierung verwendet. Die Werte werden dann über die durchgeführten Läufe gemittelt.

Es zeigt sich, dass die Sensitivität bei der Krogh-Dekodierung deutlich geringer ist. Meist beträgt sie nur etwa die Hälfte des Wertes, der bei der Viterbi-Dekodierung erzielt wird. Bei einem der Läufe trat in einigen Generationen erneut der aus Abschnitt 4.5 bekannte Fall auf, dass keine Nukleotide als zu einer CpG-Insel gehörend markiert wurden. Die Spezifität unter Verwendung des Krogh-Algorithmus liegt nach 50 Generationen über dem Wert für die Viterbi-Dekodierung.

Generationen	verpasste CpGs in %	falsche CpGs in %
0	0	91,23
10	33,33	56,11
20	33,33	62,34
30	33,33	62,98
40	35,44	88,67
50	57,81	81,13
60	57,81	47,20
70	57,81	47,20
80	46,41	65,18
90	36,71	68,75
100	24,05	72,18

Tabelle 4.7: *Verpasste und falsche CpG-Inseln für den EA bei Krogh-Dekodierung.* Für die höchstbewerteten Individuen jeder zehnten Generation des EA mit deterministischer Selektion werden basierend auf der Krogh-Dekodierung die Gütemaße bestimmt und dann jeweils über die unabhängigen Läufe gemittelt.

In Tabelle 4.7 sind zusätzlich die mittleren Werte für die verpassten und falschen CpG-Inseln dargestellt. Auch hier zeigt sich ein eher nachteiliges Verhalten bei Verwendung der Krogh-Dekodierung. Der Anteil der verpassten CpG-Inseln liegt mit Werten zwischen 24% und 58% deutlich über den Werten für die Viterbi-Dekodierung aus dem letzten Abschnitt. Der Anteil der falschen CpG-Inseln liegt bei beiden Varianten in der gleichen Größenordnung. Die prozentuale Übereinstimmung der Vorhersagen mit dem Original beträgt in der letzten Generation etwa 63%. Für die Viterbi-Dekodierung wurde ein deutlich höherer Wert von über 71% erreicht.

Speziell aufgrund der niedrigen Sensitivität und des hohen Anteils an verpassten CpG-Inseln ist die Krogh-Dekodierung für die Vorhersage der Markierungssequenzen basierend auf den vorgegebenen Ergebnis-HMMs nicht von Vorteil. Dies kann allerdings daraus resultieren, dass die HMMs zuvor unter Verwendung des Viterbi-Algorithmus angepasst und so auf seine Verwendung optimiert wurden. Wenn bereits während des Evolutionsprozesses der Krogh-Algorithmus eingesetzt werden würde, könnte sich ein völlig anderes Bild ergeben. Dies näher zu untersuchen ist jedoch aufgrund der begrenzten Rechenleistung im Rahmen dieser Arbeit nicht möglich.

Kapitel 5

Hybrider Ansatz

Abschließend soll nun in Anlehnung an memetische Algorithmen (siehe Moscato 1999) ein hybrider Ansatz, also eine Kombination aus der klassischen Anpassung mittels Baum-Welch-Algorithmus und dem zuletzt verwendeten evolutionären Algorithmus, untersucht werden. Da der Zeitaufwand für eine Iteration des Baum-Welch-Algorithmus sehr hoch ist, wird im Folgenden nur auf eines der Individuen jeder Generation eine Iteration angewendet. Dafür wird jeweils das Individuum gewählt, das aktuell den höchsten Fitnesswert besitzt. Dieses kann dann in der nächsten Generation — sofern es die deterministische Selektion überlebt — seine positiven Eigenschaften an andere Individuen weitergeben.

Bei der Auswahl des Individuums, auf das die Iteration des Baum-Welch-Algorithmus angewendet wird, ist zu beachten, dass hierfür nicht in mehreren Generationen des EAs das gleiche Individuum gewählt wird. Unter Umständen bleibt über mehrere Generationen hinweg das Individuum mit der höchsten Fitness unverändert bestehen. Das Ergebnis des Baum-Welch-Algorithmus ist deterministisch. Wird mehrmals auf das gleiche Individuum $\mathcal{I}$ jeweils eine Iteration des Baum-Welch-Algorithmus angewendet, ergibt sich jedes Mal das gleiche Individuum $\mathcal{I}^*$. Nach einer Berechnung der Fitness für dieses neue Individuum $\mathcal{I}^*$, zeigt sich, ob es in die Nachfolgegeneration übernommen wird oder nicht. Falls ja, ist es bereits in der Nachfolgegeneration enthalten und eine zweite Anwendung einer Baum-Welch-Iteration auf das Individuum $\mathcal{I}$ würde lediglich eine Kopie von $\mathcal{I}^*$ erzeugen und so zu einem Verlust der Diversität in der Population führen. Andererseits besteht die Möglichkeit, dass $\mathcal{I}^*$ nicht für die Nachfolgegeneration ausgewählt wird. Aufgrund der Verwendung der deterministischen Selektion bedeutet das, dass sein Fitnesswert zu gering ist. Auch in diesem Fall, würde es keinen Vorteil erzielen, ein zweites Mal das Individuum $\mathcal{I}$ für die Anwendung der Baum-Welch-Iteration auszuwählen. Hier wäre schon im Vorhinein klar, dass das resultierende Individuum $\mathcal{I}^*$ die Selektion für die nächste Generation erneut nicht überleben würde. Es würde also in beiden Fällen zum Einen eine Verschwendung von Rechenzeit. Zum Anderen wäre es, zumindest im ersten Fall, möglich, dass eine Population mehrfach das gleiche Individuum, nämlich das Ergebnis der einen Iteration des Baum-Welch-Algorithmus angewen-

det auf das gleiche Individuum als Ausgangspunkt, beinhaltet. Dies würde zu einem Verlust der Diversität führen und wäre ein unerwünschter Effekt. Zu diesem Zweck ist es demnach notwendig für jedes Individuum zu verwalten, ob es bereits einmal für die Anpassung mittels Baum-Welch-Algorithmus herangezogen wurde. Hierbei sei betont, dass lediglich für das ursprüngliche Individuum $\mathcal{I}$ eine neuerliche Anwendung einer Baum-Welch-Iteration ausgeschlossen wird, nicht jedoch für das Ergebnis-Individuum $\mathcal{I}^*$. Für die Anwendung der Baum-Welch-Iteration wird im Folgenden stets das Individuum der aktuellen Population gewählt, das den höchsten Fitnesswert aufweist und gleichzeitig zu keinem früheren Zeitpunkt bereits zu diesem Zweck ausgewählt wurde. Für die neuen Individuen, die durch Mutation und Rekombination entstehen, ist die zweite Bedingung natürlich trivialerweise immer erfüllt.

Es werden für den Algorithmus die gleichen Einstellungen gewählt wie für den EA in Abschnitt 4.7.5. Das heißt, dass die Populationsgröße auf sieben, die Wahrscheinlichkeit für Mutationen auf 0,25 und für Rekombinationen auf 0,75 gesetzt wird. Außerdem werden auch hier die bewährte gewichtete Fitness aus Gleichung 4.5 und die deterministische Selektion verwendet. Es werden wiederum drei unabhängige Läufe mit verschiedenen Random-Seeds gestartet und die Ergebnisse dann über die Läufe gemittelt. Die dabei erreichten Fitnesswerte sind in Abbildung 5.1 aufgetragen. Da der Zeitaufwand zur Berechnung einer Generation des hybriden Algorithmus etwa fünf Generationen des EAs entspricht, ist die Fitness in Abhängigkeit vom Zeitaufwand aufgetragen. Die Achseneinteilung beschreibt dabei die für den EA berechneten Generationen. Die komplette Länge der Achse entspricht demnach gleichzeitig 100 Generationen des EA und 20 Generationen des Hybriden. Hier fällt sofort auf, dass für den hybriden Algorithmus von Anfang an höhere Fitnesswerte erreicht werden und diese im weiteren Verlauf ebenfalls schneller ansteigen. Die für den Hybriden erreichten Fitnesswerte liegen noch einmal um etwa 10% höher als bei dem zuletzt untersuchten EA.

Während der Läufe wird protokolliert, welche der mit dem Baum-Welch-Algorithmus bearbeiteten Individuen die Selektion für die Nachfolgegeneration überleben. In über der Hälfte der Generationen wurden diese Individuen selektiert. Dies zeigt, dass die zusätzlich investierte Rechenzeit meist auch Auswirkungen auf die nachfolgenden Generationen hat, da die Individuen für die Erzeugung von Nachfolgeindividuen zur Verfügung stehen. Bei der Protokollierung der selektierten Individuen fällt ebenfalls auf, dass in frühen Generationen die Mehrheit der durch Mutation, Rekombination und Baum-Welch-Algorithmus neu erzeugten Individuen den Sprung in die Nachfolgepopulation schaffen. Mit steigender Generationenzahl sinkt der Anteil an erfolgreichen Mutationen und Rekombinationen. Bemerkenswert ist jedoch, dass dies nicht für die durch die Anwendung der Baum-Welch-Iteration erzeugten Individuen gilt: Auch in späteren Generationen schaffen diese noch genauso oft den Sprung in die Nachfolgepopulation wie in den anfänglichen Generationen. Ihre Erfolgsrate bleibt annähernd gleich. Daher bietet die zusätzlich durchgeführte Baum-Welch-Iteration auch dann noch

eine Möglichkeit der Verbesserung des Ergebnisses, wenn durch einen rein evolutionären Algorithmus keine oder nur noch wenige Individuen mit einem höheren Fitnesswert erzeugt würden.

Um zu überprüfen, ob die Individuen mit diesen hohen Fitnesswerten tatsächlich bessere Vorhersagen treffen, werden für im Folgenden ebenfalls die Gütemaße bestimmt. Die durchschnittlichen Sensitivitäts- und Spezifitätswerte, die für die verschiedenen Random-Seeds ermittelt wurden, sind in Abbildung 5.2 zusammen mit den Werten für den reinen EA aufgetragen. Als Skala dient hier, wie bereits in Abbildung 5.1, die Zeit, in der eine bestimmte Anzahl an Generationen des EAs berechnet werden kann. Eine Generation des Hybriden entspricht dabei vom Zeitaufwand fünf Generationen des EA. Für Ergebnisse des hybriden Algorithmus wird auf den Nachbearbeitungsschritt verzichtet, da er in diesem Fall kaum einen Effekt auf die Gütemaße zeigt.

Wie Abbildung 5.2 deutlich zeigt, ist die Sensitivität für den hybriden Algorithmus nach gleicher Rechenzeit durchweg höher als für den EA. Auch für die Spezifität zeigt der hybride Ansatz schnell ähnliche, teils bessere Werte. Dies spiegelt sich auch in Tabelle 5.1 wider, wo die durchschnittlichen Werte für die verpassten und falschen CpG-Inseln aufgetragen sind. Da hier nur die Werte für den Hybriden dargestellt sind, dient in diesem Fall die Anzahl der für den Hybriden berechneten Generationen als Skala. Um die Werte mit denen für den EA aus Tabelle 4.6 auf Seite 71 zu vergleichen, muss beachtet werden, dass die Berechnung einer Generation des hybriden Algorithmus etwa genauso viel Zeit benötigt wie die von fünf Generationen des reinen EA. Hier zeigt sich, dass für den Hybriden deutlich weniger tatsächliche CpG-Inseln verpasst, also nicht von einer vorhergesagten überlappt, werden. Der Anteil an falschen CpG-Inseln ist bei dem hybriden Ansatz ebenfalls geringer. Diese Werte sind geringfügig besser als diejenigen, die für den Baum-Welch-Algorithmus ermittelt wurden.

Generationen	verpasste CpGs in %	falsche CpGs in %
0	0	92,98
5	2,53	64,49
10	2,95	60,71
15	2,95	60,51
20	2,11	59,70

Tabelle 5.1: *Verpasste und falsche CpG-Inseln für den hybriden Algorithmus.* Die Werte für die höchstbewerteten Individuen jeder fünften Generation des Hybriden in den einzelnen Läufen werden über die unabhängigen Läufe gemittelt.

Abbildung 5.3 stellt die gemittelten Werte für die Sensitivität und die Spezifität der Ergebnisse des hybriden Algorithmus für die beiden Dekodierungsvarianten im direkten Vergleich dar. Es zeigt sich, dass durch die Dekodierung mittels Krogh-Algorithmus kei-

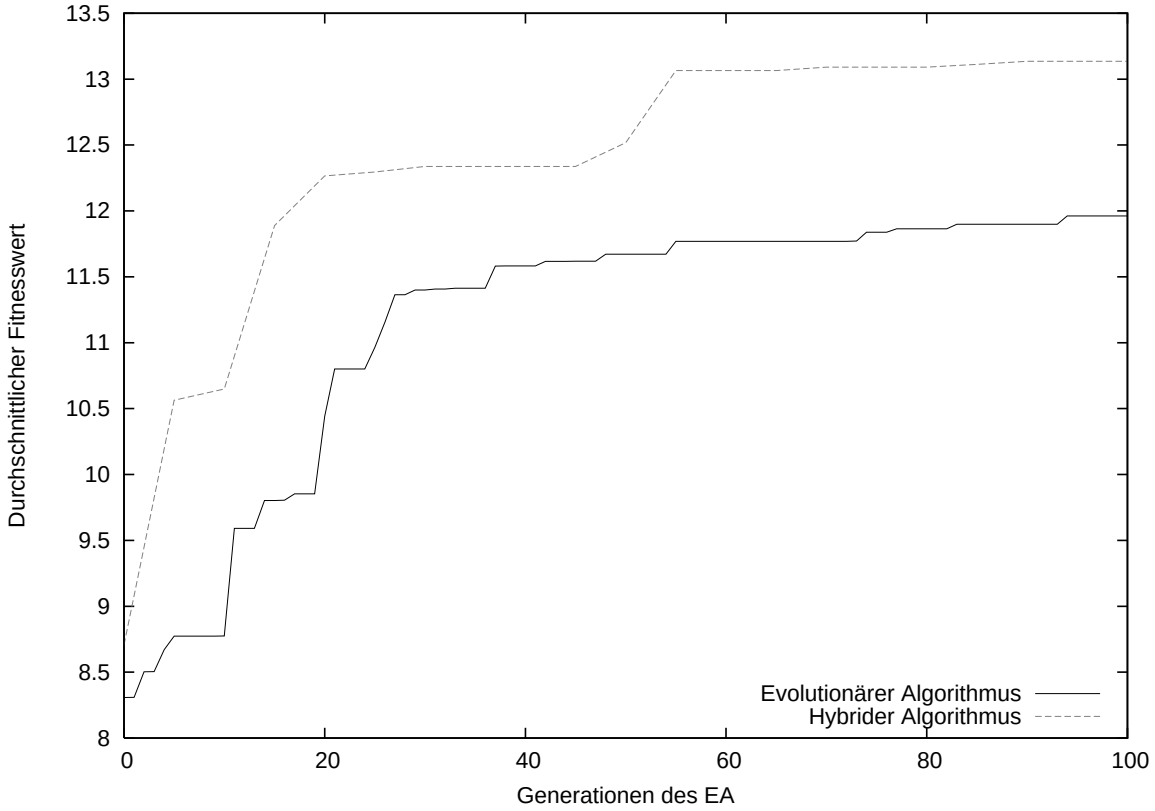


Abbildung 5.1: *Vergleich der durchschnittlichen Fitnesswerte für EA und hybriden Algorithmus.* Die maximalen Fitnesswerte der einzelnen Läufe des EA und des hybriden Algorithmus werden über die einzelnen unabhängigen Läufe gemittelt. Als Skala werden hier die Generationen des EA gewählt. Eine Generation des Hybriden entspricht dabei fünf Generationen des EA. Zum Vergleich weist das nach 80 Iterationen des Baum-Welch-Algorithmus resultierende HMM eine Fitness von 13,56 auf.

ne wesentlich besseren Ergebnisse erreicht werden. Die Sensitivität ist geringer als bei der Viterbi-Dekodierung, während die Spezifität beim Krogh-Algorithmus etwas höher ist. Da jedoch das Ziel ist, die CpG-Inseln zu identifizieren, ist es wichtiger die CpG-Inseln überhaupt abzudecken. Dass dabei mehr Nukleotide als zu CpG-Inseln gehörend markiert werden, ist dem Fall von fälschlicherweise als nicht in einer CpG-Insel liegende markierten Nukleotiden vorzuziehen. Da der Krogh-Algorithmus außerdem mit einer wesentlich höheren Rechenzeit einhergeht, stellt sich der Viterbi-Algorithmus an dieser Stelle ebenfalls als vorteilhaftere Variante dar.

Abschließend sind in den Abbildungen 5.4 und 5.5 für zwei Sequenzen der Validierungsdaten die Ergebnisse der verschiedenen Methoden zusammengefasst. Hier handelt es sich um die gleichen beiden Sequenzen wie in den Abbildungen 4.20 und 4.21. Zusätz-

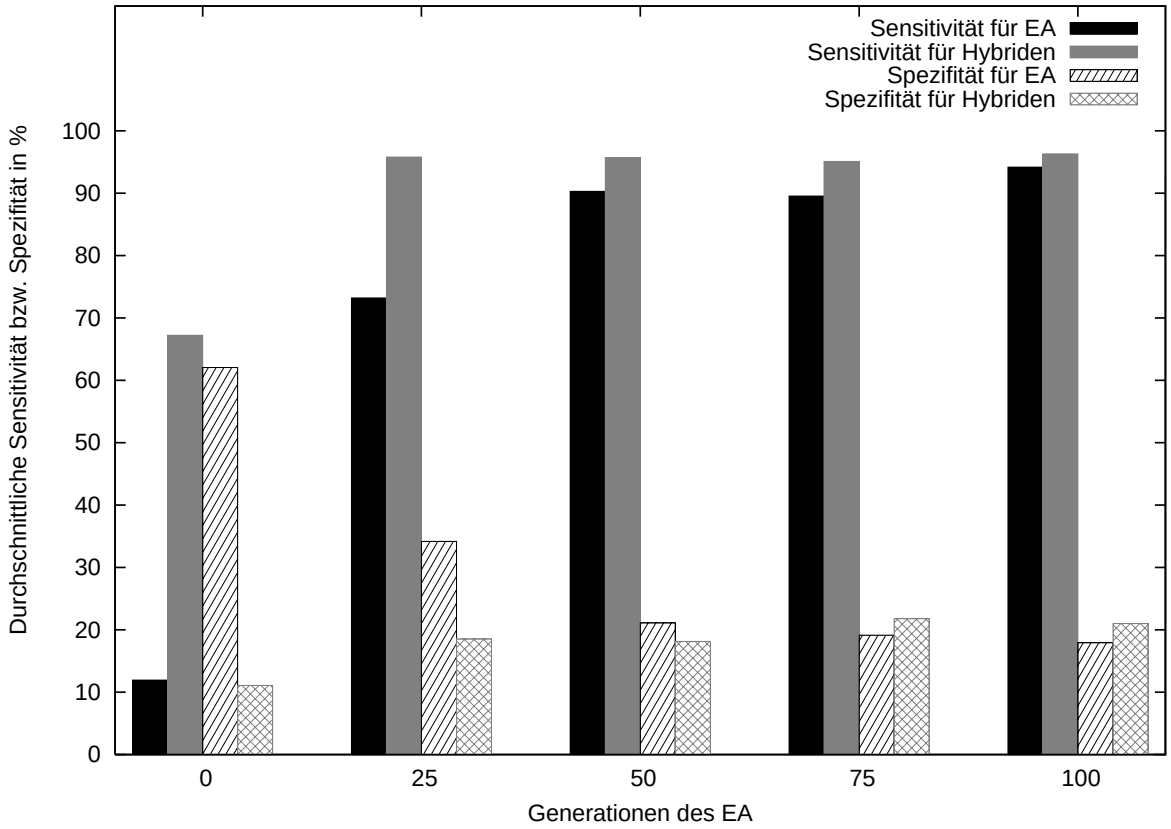


Abbildung 5.2: *Sensitivität und Spezifität für EA und Hybriden im Vergleich.* Die Werte für den EA entsprechen über die Läufe gemittelten Werten nach Anwendung des Nachbearbeitungsschritts. Für den Hybriden werden nach jeder fünften Generation für die drei unabhängigen Läufe jeweils die Gütemaße auf Grundlage der Viterbi-Dekodierung bestimmt und ebenfalls gemittelt.

lich zu den dort schon abgebildeten Ergebnissen nach Anwendung des Baum-Welch-Algorithmus und des EAs sind hier die Vorhersagen des hybriden Algorithmus zu sehen. Wie bereits aufgrund der Abbildungen 5.1 bis 5.3 erwartet, erweist sich der hybride Algorithmus als die beste Alternative. Es werden prozentuale Übereinstimmungen mit den vorgegebenen Daten um 80% erreicht. Obwohl dies den Ergebnissen, die durch die reine Anwendung des Baum-Welch-Algorithmus erzielt wurden, entspricht, erreicht der hybride Algorithmus schon nach deutlich kürzerer Zeit diese sehr guten Ergebnisse. Außerdem weisen die Ergebnisse des Hybriden eine klarere Tendenz zu einer besseren Anpassung des HMMs auf, während die Gütemaße für den Baum-Welch-Algorithmus zum Teil stark schwankten. Dies liegt unter Umständen darin begründet, dass der Baum-Welch-Algorithmus nicht für Sequenzen mit Markierungen ausgelegt ist und somit nicht in der Lage ist, die CML zu maximieren.

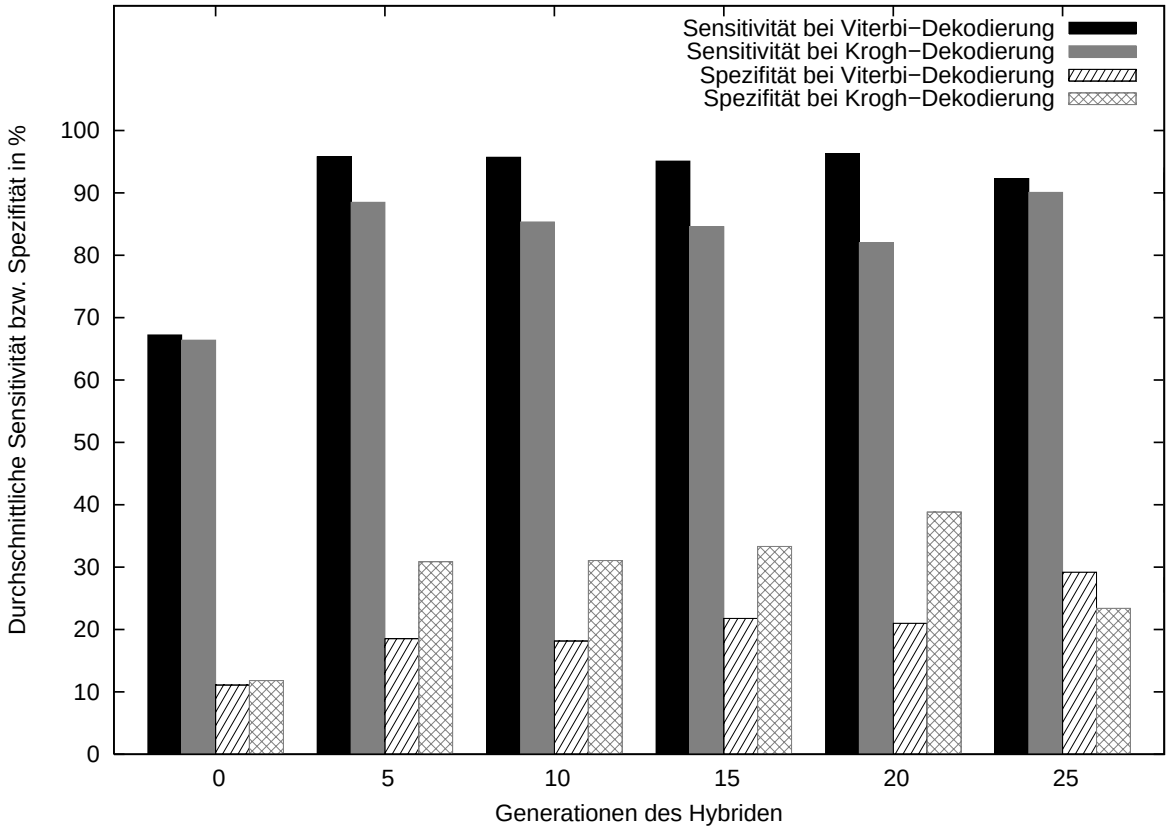


Abbildung 5.3: *Vergleich zwischen Viterbi- und Krogh-Dekodierung auf den Ergebnissen des hybriden Algorithmus.* Nach jeder fünften Generation des Hybriden werden für das höchstbewertete Individuum jeweils einmal auf Basis der Viterbi-Dekodierung und einmal auf Basis der Krogh-Dekodierung die Gütemaße bestimmmt und über die unabhängigen Läufe gemittelt aufgetragen.

Abbildung 5.4: *Vergleich der Markierungen bei den verschiedenen Methoden für eine Sequenz der Validierungsdaten.* Zusätzlich zur Originalsequenz, dem Ergebnis des Baum-Welch-Algorithmus nach 80 Iterationen und einem Ergebnis-HMM des EAs mit deterministischer Selektion nach 100 Generationen ist hier das Ergebnis des Hybriden nach 20 Generationen dargestellt. CpG-Inseln sind weiß, die übrigen Regionen schwarz eingefärbt.

Um die Tendenz des hybriden Algorithmus zu besseren Ergebnissen zu verifizieren, werden für die drei unabhängigen Läufe jeweils 15 zusätzliche Generationen berechnet. Dies entspricht einer Laufzeit von 150 Generationen des EA bzw. von gut 40 Iterationen des Baum-Welch Algorithmus. Nach insgesamt 35 Generationen des hybriden Algorithmus liegt die Fitness der Individuen bereits bei 13,74. Damit ist die Bewertung durch die Fitness höher als nach 80 Iterationen des Baum-Welch-Algorithmus. Dort wurde eine Fitness von 13,56 erreicht. Die Individuen des Hybriden nach 35 Generationen entsprechen einer Sensitivität von 93,47% bei einer Spezifität von 43,43%. Der Anteil an verpassten CpG-Inseln beträgt 3,80%, während der Anteil an falschen CpG-Inseln lediglich noch bei 35,71% liegt. Die Vorhersagen der Markierungen auf den Validierungsdaten stimmen dabei bereits zu 92,45% mit den vorgegebenen Daten überein. Dies übertrifft die Ergebnisse aller zuvor in dieser Arbeit getesteten Vorhersagemethoden.

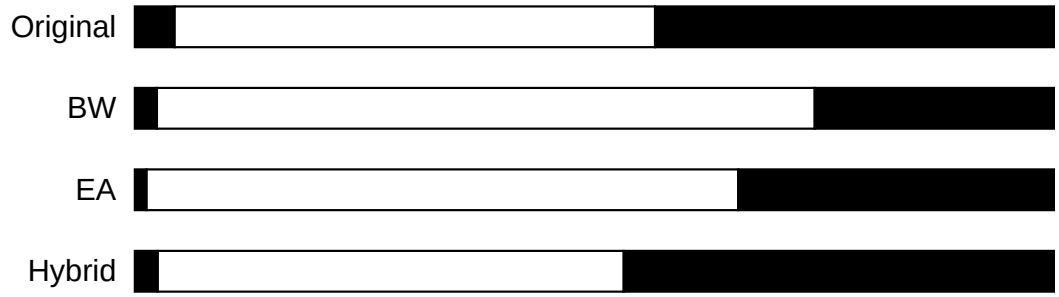


Abbildung 5.5: *Vergleich der Markierungen bei den verschiedenen Methoden für eine zweite Sequenz der Validierungsdaten.* Zusätzlich zu den Sequenzen aus Abbildung 4.21 (Originalsequenz, Ergebnis des Baum-Welch-Algorithmus nach 80 Iterationen, Ergebnis-HMM des EAs mit deterministischer Selektion nach 100 Generationen und Nachbearbeitungsschritt), ist hier das Ergebnis des hybriden Algorithmus nach 20 Generationen visualisiert. CpG-Inseln sind weiß dargestellt, während die übrigen Regionen schwarz eingefärbt sind.

Kapitel 6

Zusammenfassung und Ausblick

Diese Diplomarbeit befasst sich mit Methoden zur Identifizierung von CpG-Inseln in DNA-Sequenzen. Dazu wurden verschiedene Ansätze zur Anpassung von Hidden Markov Modellen untersucht, entwickelt und anhand von Gütemaßen miteinander verglichen. Zunächst wurde die verbreitetste Methode für die Anpassung von HMMs, der Baum-Welch-Algorithmus, betrachtet. Hierbei fiel auf, dass der Baum-Welch Algorithmus zwar in der Lage ist, die Maximum Likelihood, also die Wahrscheinlichkeit, dass eine oder mehrere Sequenzen von dem angepassten HMM erzeugt wurden, zu maximieren, dies allerdings für die hier betrachtete Problemstellung nicht dem eigentlichen Ziel entspricht. Bei der Identifizierung von CpG-Inseln, wie auch bei allen anderen Problemstellungen bei denen Markierungen zu den Ausgabesequenzen berücksichtigt werden müssen, geht es darum, eine möglichst gute Vorhersage für die Markierungen entlang unbekannter Sequenzen zu treffen. Für die Verbesserung der Vorhersagen war allerdings beim Baum-Welch-Algorithmus keine klare Tendenz erkennbar.

Durch den Baum-Welch-Algorithmus wurde nach 80 Iterationen auf unbekannten Testdaten eine Übereinstimmung der Vorhersage mit dem Original von 80,08% erzielt. Da die Anzahl korrekt klassifizierter Nukleotide in den Iterationen des Baum-Welch-Algorithmus nicht monoton ist, wurden zu früheren Zeitpunkten höhere Übereinstimmungen erreicht, die zugehörigen HMMs allerdings im Verlauf des Algorithmus wieder verworfen. Eine klare Verbesserung der Vorhersagen ließ sich nicht erkennen. Gleichzeitig wurde der Nachteil der hohen Laufzeit des Baum-Welch-Algorithmus deutlich.

Um diesen Problemen zu begegnen, wurden evolutionäre Ansätze als Alternative zum Baum-Welch-Algorithmus verwendet. Bei deren Entwicklung traten zunächst einige Schwierigkeiten mit den verwendeten Fitnessfunktionen auf. Da die in der Literatur vorgeschlagene Fitnessfunktion auf Basis der Maximum Likelihood ebenfalls eine sehr hohe Laufzeit aufweist und die Markierungen nicht mit einbezieht, wurden verschiedene Alternativen untersucht, dies durch geeignetere Fitnessfunktionen zu beheben. Die Realisierung dieser Ziele geschah schließlich in Form einer Funktion, die die Übereinstimmung der Vorhersage und der vorgegebenen Sequenz misst und diese nach der Art

der Übereinstimmung gewichtet. Die Gewichtung teilte dabei in CpG-Inseln liegenden Nukleotiden größere Bedeutung zu als außerhalb liegenden, da diese in DNA-Sequenzen seltener vorkommen. Ohne diese Gewichtung wurde die Fitness durch das häufigere Vorkommen der Markierung, die eine Region außerhalb von CpG-Inseln anzeigt, dominiert.

Außerdem zeigte sich im Verlauf der Entwicklung, dass die zunächst gewählte Rouletteradselektion nicht den Anforderungen entsprach, so dass sie erst modifiziert und schließlich durch eine deterministische Selektion der Individuen mit der höchsten Fitness ersetzt wurde.

Für die verschiedenen Ansätze, die im Laufe dieser Entwicklung entstanden, wurden für Parameter (Populationsgröße, Mutations- und Rekombinationswahrscheinlichkeit) mithilfe der sequentiellen Parameteroptimierung möglichst gute Einstellungen ermittelt. So mussten die Parameter nicht durch Ausprobieren festgelegt werden, sondern es wurden durch ein analytisches Verfahren besonders vorteilhafte Parameter für den EA mit der jeweiligen Fitnessfunktion berechnet.

Die auf diese Art gewonnenen Ergebnisse lagen zwar mit einer Übereinstimmung von 71,39% hinter den Ergebnissen des Baum-Welch-Algorithmus zurück, allerdings wurde hierfür weniger als ein Drittel der Zeit benötigt.

Da die Möglichkeiten der reinen EAs erschöpft schienen, wurde schließlich ein hybrider Ansatz entwickelt. Dieser sollte eine Verbindung zwischen klassischem Ansatz in Form des Baum-Welch-Algorithmus und den evolutionären Algorithmen schaffen. Aufgrund des großen Zeitaufwands des Baum-Welch-Algorithmus im Vergleich zu dem des evolutionären Ansatzes, musste hier ein Kompromiss gefunden werden. Dies geschah, indem nach jeder Generation des EA auf das jeweils am höchsten bewertete Individuum eine Iteration des Baum-Welch-Algorithmus angewendet wurde. Dieses Individuum wurde dann ebenfalls in die Selektion mit einbezogen.

Durch den hybriden Ansatz wurden Ergebnisse erzielt, die sowohl die des Baum-Welch- als auch des evolutionären Algorithmus übertrafen. So wurde bereits nach 35 Generationen eine Übereinstimmung von 92,45% erreicht. Dies entspricht etwa der Hälfte der Zeit, die für die Berechnungen mit dem Baum-Welch-Algorithmus verwendet wurde. Auch bei dem reinen EA wurden in gleicher Zeit keine ähnlich guten Ergebnisse erzielt.

Weitere Untersuchungen auf diesem Gebiet erscheinen vielversprechend. Dabei gibt es eine Vielzahl von Aspekten, an denen weitergearbeitet werden sollte. Hierbei wäre es zunächst interessant, einen dem Baum-Welch-Algorithmus entsprechenden Algorithmus zu finden, der anstelle der ML die CML maximiert. Da problemspezifische Ansätze meist effizienter sind als EAs, würde eine Berücksichtigung der Markierungen der Trai-

ningsdaten ohne die bisher zu diesem Zweck eingesetzten evolutionären Algorithmen erzielt werden.

Zusätzlich zu den durchgeführten Untersuchungen könnten die einzelnen Mutations- und Rekombinationsoperatoren untersucht werden. Wenn sich dabei einzelne Operatoren als besonders vorteilhaft erweisen, könnten sie mit einer höheren Wahrscheinlichkeit gewählt werden als die anderen. Außerdem könnte, nach dem Vorbild der von Rechenberg vorgestellten 1/5-Erfolgsregel, eine Anpassung der Schrittweiten erfolgen, sobald der Anstieg der Fitness absackt oder wenn nur noch sehr wenige der neu erzeugten Individuen für die Nachfolgepopulation ausgewählt werden. Dies würde der Vermutung entsprechen, dass die Nähe zum Optimum im Suchraum dazu führt, dass nur noch wenige Mutationen und Rekombinationen erfolgreich sind.

Aufgrund der beschränkten, im Rahmen dieser Arbeit zur Verfügung stehenden Rechenkapazität konnten hier einige Ansätze nicht weiter verfolgt werden. Hierzu zählt beispielsweise die Verwendung der Krogh-Dekodierung bei der Berechnung der Fitnessfunktion. Bisher hatte sich der Nutzen der Krogh-Dekodierung für die Markierungsvorhersage auf den Validierungsdaten zwar als eher gering dargestellt, allerdings beruhte die Bewertung der Individuen während des Optimierungsprozesses des EAs ausschließlich auf der Viterbi-Dekodierung. Wenn bereits für die Fitnessberechnung der Algorithmus von Krogh miteinbezogen werden würde, würde dies vermutlich anders aussehen.

Außerdem würde mehr Rechenkapazität eine Ausweitung des hybriden Algorithmus zulassen. Es könnte nicht — wie hier geschehen — nur für ein Individuum jeder Generation eine Iteration des Baum-Welch-Algorithmus angewendet werden, sondern beispielsweise auf jedes, für das dies nicht bereits in einer früheren Generation geschehen ist. Außerdem besteht die Möglichkeit, zwei oder mehr Iterationen des Baum-Welch-Algorithmus für die einzelnen Individuen zu berechnen. Dabei wäre es interessant zu sehen, ob der zusätzliche Rechenaufwand eine genauere Anpassung an die Problemstellung ermöglicht. Weiterhin wäre eine Anpassung an die Evolution denkbar: Wird über mehrere Generationen hinweg ein nur noch minimaler Anstieg der Fitness erzielt, könnte die Anzahl der Baum-Welch-Iterationen erhöht oder die Anzahl der Individuen, auf die diese Iterationen angewendet werden, gesteigert werden.

Die Miteinbeziehung der CML in die Fitness war aufgrund der hohen Rechenzeit hier ebenfalls nicht möglich, könnte aber in der Zukunft mit fortschreitender Technologie und schnelleren Rechnern durchaus wieder aufgegriffen werden.

Schließlich wäre es noch interessant, die hier entwickelten Ansätze, speziell den hybriden Algorithmus, auf andere Probleme in der Bioinformatik und auf weitere Trainingsdaten anzuwenden und zu überprüfen, ob dort ähnlich gute Ergebnisse erzielt werden können.

Zusammenfassend kann gesagt werden, dass die Fitnessfunktion maßgeblich das Verhalten eines evolutionären Algorithmus bestimmt. Eine adäquate Fitnessfunktion ist damit wesentlich entscheidender als die Parametereinstellung von Populationsgröße und den Wahrscheinlichkeiten für Mutation und Rekombination. Solange die Fitnessfunktion nicht genau das gewünschte Verhalten widerspiegelt, können auch vermeintlich günstige Parametereinstellungen nicht das gewünschte Maß an Verbesserungen erzielen.

Die in der Einleitung gestellte Frage nach dem Nutzen einer Kombination der klassischen Optimierung mittels Baum-Welch-Algorithmus und evolutionären Algorithmen kann klar bejaht werden: Durch den hybriden Algorithmus werden Ergebnisse erzielt, die die Ergebnisse der einzelnen Verfahren übertreffen. Die Aufgabe zukünftiger Untersuchungen ist es nun, das vorhandene Potential weiter auszuschöpfen.

Literaturverzeichnis

Bartz-Beielstein 2006
Bartz–Beielstein, Thomas: *Experimental Research in Evolutionary Computation – The New Experimentalism.* Berlin: Springer, 2006, (Natural Computing Series).

Baum 1972
Baum, Leonard E.: An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes. In: *Inequalities III. Proceedings of the 3rd Symposium on Inequalities*, S. 1–8. Hrsg. : Shisha, Oved. New York: Academic Press, 1972.

Durbin u. a. 1998
Durbin, Richard; Eddy, Sean R.; Krogh, Anders; Mitchison, Graeme: *Biological Sequence Analysis. Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

Fahrmeir u. a. 2003
Fahrmeir, Ludwig; Künstler, Rita; Pigeot, Iris; Tutz, Gerhard : *Statistik – Der Weg zur Datenanalyse*, 4. Auflage. Berlin: Springer, 2003.

Koski 2001
Koski, Timo: *Hidden Markov Models for Bioinformatics.* Dordrecht, Niederlande: Kluwer Academic Publishers, 2001.

Krogh 1994
Krogh, Anders: Hidden Markov Models for Labeled Sequences. In: *Proceedings of the 12th International Association on Pattern Recognition (IAPR)*, S. 140–144. Hrsg. : Butler, Jon T.; Chang, Carl K.; Rine, David C.; Singhal, Muskesh; Srimani, Pradip K.; Williams, Michael Roy. Los Alamitos, CA, USA: IEEE Press, 1994.

Krogh 1997
Krogh, Anders: Two Methods for Improving Performance of a HMM and Their Application for Gene Finding. In: *Proceedings of the 5th International Conference*

on Intelligent Systems in Molecular Biology (ISMB), S. 179–186. Hrsg. : Gaasterland, Terry; Karp, Peter; Karplus Kevin; Ouzounis, Christos; Sander, Chris; Valencia, Alfonso. Menlo Park, CA, USA: AAAI Press, 1997.

McKay u. a. 1979
McKay, Michael D.; Beckman, Richard J.; Conover, William J.: A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. In: *Technometrics,* Band 21, S. 239–245. Alexandria, VA, USA: American Society for Quality and the American Statistical Association, 1997.

Moscato 1999
Moscato, Pablo: Memetic Algorithms: A Short Introduction. In: *New Ideas in Optimization*, S. 219—234. Hrsg. : Corne, David; Dasgupta, Dipankar; Dorigo, Marco; Glover, Fred; Moscato, Pablo; Poli Riccardo; Price, Kenneth. Maidenhead, UK, England: McGraw-Hill, 1999.

Rabiner 1989
Rabiner, Lawrence R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In: *Proceedings of the IEEE 77,* Band 2, S. 257–286. Hrsg. : Ulaby, Fawwaz T.; Institute of Electrical and Electronics Engineers. Los Alamitos, CA, USA: IEEE Press, 1989.

Rechenberg 1973
Rechenberg, Ingo: *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Stuttgart: Frommann-Holzboog, 1973.

Schwartz und Chow 1990
Schwartz, Richard; Chow, Yen–Lu: The N-best Algorithm: An Efficient and Exact Procedure for Finding the N Most Likely Hypotheses. In: *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP),* S. 81–84. Los Alamitos, CA, USA: IEEE Press, 1990.

Schwefel 1975
Schwefel, Hans–Paul: *Evolutionsstrategie und numerische Optimierung.* Dr.-Ing. Dissertation, Technische Universität Berlin, Fachbereich Verfahrenstechnik, 1975.

Thomsen 2002
Thomsen, Renè: Evolving the Topology of Hidden Markov Models Using Evolutionary Algorithms. In: *Proceedings of the 7th Conference on Parallel Problem Solving from Nature (PPSN)*, LNCS 2439, S. 861-870. Hrsg. : Merelo, Juan J.; Panagiotis, Adamidis; Beyer, Hans-Georg; Fernández-Villaca nas, José-Luis;Schwefel, Hans-Paul. Berlin Heidelberg: Springer Verlag, 2002.

Won u. a. 2005

Won, Kyoung-Jae; Hamelryck, Thomas; Prügel-Bennett, Adam; Krogh, Anders: Evolving Hidden Markov Models for Protein Secondary Structure Prediction. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, Band 1, S. 33-40. Hrsg. : Corne, David; Michalewicz, Zbigniew; McKay, Bob; Eiben, Gusz; Fogel, David; Fonseca, Carlos; Greenwood, Garrison; Raidl, Gunther; Tan, Kay Chen; Zalzala, Ali. Edinburgh, Scotland, UK: IEEE Press, 2005.

Anhang A

Implementierung

Der für die vorliegende Arbeit implementierte MATLAB-Code ist auf der beiliegenden CD enthalten. Dieser umfasst mehrere Skripte zur Ausführung der einzelnen untersuchten Algorithmen.

Das Skript `startRun` führt einen Lauf des Baum-Welch-Algorithmus aus. In diesem kann eine Datei angegeben werden, in der zusätzliche Konfigurationen (Anzahl Iterationen, Dateinamen für die Ausgaben, etc.) vorgenommen werden können. Die Datei `bwOptions.m` enthält ein Beispiel dazu und zeigt, welche Parameter belegt werden können. Wird kein Dateiname angegeben, werden die in `getBWoptions` spezifizierten Standardwerte verwendet.

Ein Lauf des evolutionären oder hybriden Algorithmus kann mithilfe des Skripts `startEA` gestartet werden. Ein Lauf des hybriden Algorithmus stellt dabei eine Erweiterung des evolutionären Algorithmus dar. Dazu muss lediglich der entsprechende Parameter auf 1 gesetzt werden. Das Skript verwendet, wenn nicht anders angegeben, die Standardwerte aus `getDefaultOptions`. In den Optionen steht eine Vielzahl von Einstellungen zur Verfügung. Dort kann unter anderem unter allen vier in dieser Arbeit betrachteten Fitnessfunktionen und den zwei Selektionsvarianten gewählt werden. Außerdem wird dort festgelegt, ob ein reiner EA oder der hybride Algorithmus verwendet werden soll. Alternativ kann, um andere Optionen zu verwenden, die Funktion `getUserOptions` eingesetzt werden. Dieser Funktion wird der Name der Datei übergeben, die die zu verwendenden Optionen enthält. Die Funktion gibt ein Objekt, das die gewünschten Optionen enthält, zurück und dieses wird dann wiederum der Funktion `startEARun` übergeben, die dann schließlich den Lauf startet.

Zur Verwendung in Kombination mit der SPO-Toolbox[1], die im Begleitmaterial zu Bartz–Beielstein (2006) enthalten ist, steht die Funktion `SPOTRun` zur Verfügung. Diese verwendet die in `getSpotOptions` angegebenen Optionen. Hier können ebenfalls

[1]Die SPO-Toolbox steht zusammen mit ihrer Dokumentation unter der URL `http://ls11-www.cs.uni-dortmund.de/people/tom/ExperimentalResearchPrograms.html` zum Download bereit.

Fitnessfunktion, Selektionsoperator und Art des Algorithmus festgelegt werden. Außerdem besteht die Möglichkeit, die Berechnung des Funktionswertes Y für einen einzelnen SPO-Designpunkt aufzurufen. Dadurch kann die Berechnung der Funktionswerte für die Designpunkte auf mehreren Rechnern parallelisiert werden. Hierzu wird die Funktion `singleSPOTRun` aufgerufen, die ebenfalls die in `getSpotOptions` enthaltenen Optionen verwendet. Dieser Funktion müssen der SPO-Designname (Bezeichnung der Datei, die das SPO-Design enthält), der Designindex sowie der Index der Wiederholung des gewünschten Laufs, der indirekt den Random-Seed bestimmt, übergeben werden.

Neben den in dieser Arbeit bereits verwendeten Funktionen enthält die CD weitere Möglichkeiten, die über Optionen in den jeweiligen Optionsdateien gesteuert werden können und so für zusätziche Untersuchungen verwendet werden können. Als solches sei beispielsweise erwähnt, dass die Möglichkeit zur Steuerung der Anzahl Baum-Welch-Iterationen beim hybriden Algorithmus sowie eine auf der CML basierende Fitnessfunktion bereits implementiert wurden, allerdings aufgrund der hohen Laufzeit auf aktuellen Rechnern nicht in adäquater Zeit ausgeführt werden konnten und daher nicht näher untersucht wurden.

Algorithmenverzeichnis

Abbildungsverzeichnis