UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Bachelorarbeit

Fancy thesis template

Vorname Nachname
Matrikelnummer: 0123456
Angewandte Informatik (Bachelor)

Fachgebiet Verteilte Systeme, Abteilung Informatik
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

14. März 2023

Erstgutachter: Prof. Dr-Ing. Torben Weis
Zweitgutachter: Hier Zweitgutachter eintragen
Zeitraum: 1. September 2042 - 1. Januar 2043

Abstract

Hidden Markov Modelle sein ein weit verbreitetes Werkzeug für Signalmodellierung. Die Parameter eines HMM werden typischerweise mit dem Baum-Welch Algorithmus trainiert, welcher jedoch nur lokale Optima findet. Der Baum-Welch Algorithmus wurde zu Begin der 1970er Jahre entwickelt und ist somit über 50 Jahre alt. In dieser Zeit wurden viele alternative Trainingsverfahren entwickelt mit dem Ziel eine bessere globale Suchkapazität zu bieten. Einige dieser Algorithmen sind sogenannte Metaheuristiken welche oft den Baum-Welch Algorithmus hybridisieren. Ziel dieser Arbeit ist es zu evaluieren in wie weit solche (hybriden) Metaheuristiken geeignet sind für die Parameter-Estimation eines HMM, wobei wir uns in der Auswertung auf genetische Algorithmen beschränken, welche zu den meißterforschten Metaheuristiken zählen. Zu diesem Zweck wurde ein Framework für das trainieren von HMMs mit genetischen Algorithmen entwickelt, mit welchem drei verschiedene Ansätze untersucht werden. Die Auswertung zeigt, dass genetische Algorithmen nicht geeignet sind für das Trainieren von HMMs und das der Baum-Welch Algorithmus nicht ohne Grund weiterhin der Standard für die Parameter-Estimation ist.

Contents

Chapter 1

Introduction

Hidden Markov Modelle bieten die möglichkeit komplexe Systeme zu modellieren. Sie wurden im frühen zwanzigsten Jahrhundert durch Andrey Markov eingeführt [**rabiner**]. Zunächst war die Forschung an Hidden Markov Modellen primär theoretischer Natur, bis in den späten 1960er bis frühen 1970er Jahren effiziente Methoden für die Parameterestimation durch Leonard E. Baum und seine Kollegen entwickelt wurden. Seit jeher finden Hidden Markov Modelle praktische Anwendungen in einer Vielzahl an Wissenschaftlichen Disziplinen. Hidden Markov Modelle sind beliebt, da sie unter anderem sehr flexibel und einfach zu implementieren sind [**HmmReview**] und mittleerweile aus den Gebieten der Spracherkennung [**ApplicationSpeechRecognition**] und Bioinformatik [**ApplicationComputationalBiology**] nicht mehr wegzudenken. Eine der prominentesten Anwendungen von Hidden Markov Modellen ist der Google Pagerank Algorithmus, welcher die Relevanz von Webseite in Relation zu einer Suchanfrage ermittelt [**ApplicationPageRank**].

Motivation

Wie gut ein trainiertes Hidden Markov Model einen Prozess modelliert ist abhängig von der Qualität der Parameter des Modells. Die am häufigsten verwendete Methode für das ermitteln der Parameter ist der Baum-Welch Algorithmus. Dieser ist jedoch anfällig dafür in lokalen Optima stecken zu bleiben. Daher wurden in den letzten Jahren zahlreiche alternative Trainingsverfahren für HMMs entwickelt. Viele davon sind sogenannte Metaheuristiken, stochastische Verfahren die durch natürliche Prozesse wie Evolution inspiriert sind. Trotz dieser großen Auswahl alternativer Trainingsverfahren besteht die predominante Trainingsstrategie weiterhin darin, den Baum-Welch Algorithmus mehrmals auf verschiedene initiale Parametern anzuwenden und das beste resultierende Modell zu wählen. So lautet es zumindestens in der Dokumentation zu *hmmlearn*, einer der beliebtesten HMM python libraries. Es stellt sich also die Frage warum keine der alternativen Trainingsverfahren so großen Anklang wie der Baum-Welch Algorithmus gefunden hat.

Struktur

Ich werde zunächst die Grundlagen auslegen, dazu gehören eine Übersicht zum Aufbau und stochastische Methoden für Hidden Markov Modelle. Danach widmen wir uns allgemeiner dem Gebiet der Optimierung und gehe auf Metaheuristische Algorithmen ein sowie derer Kritik. Danach werden genetische Algorithmen erklärt. Im zweiten Kapitel werden die Konzepte aus den Grundlagen kombiniert und es wird darauf eingegangen wie man Hidden Markov Modelle mit einem genetischen Algorithmus kombinieren kann und worauf man dabei achten sollte. Des Weiteren wird ein Überblick verschafft über die bisher unternommenen metaheuristischen Trainingsansätze für HMMs. Im dritten Kapitel gehe ich auf die Implementation eines Frameworks für genetische Algorithmen und Hidden Markov Modelle ein. Danach werden drei Ausgewählte Ansätze die Hidden Markov Modelle mit einem genetischen Algorithmus trainieren evaluiert. Die Arbeit konkludiert mit einem Fazit über die feasibility von genetischen Algorithmen für das training von HMMs, so wie einer Diskussion über den Zustand des Feldes der Metaheuristischen Optimierung und gibt einen Ausblick über zukünftige Forschung.

Chapter 2

Grundlagen

2.1 Grundbegriffe der Stochastik

Zufallsexperimente und Elementarereignisse

Der Grundbaustein der Stochastik sind sogenannte **Zufallsexperimente**. Klassische Beispiele für Zufallsexperimente sind das Werfen eines fairen 6-seitigen Wüfels oder das Ziehen von Kugeln aus einer Urne. Das Ergebnis eines Zufallsexperimentes nennt man **Elementarereignis** w. Die Menge aller Möglichen Elementarereignisse ist Ω. Wenn unser Zufallsexperiment das einmalige Werfen eines fairen Würfels ist, dann gilt $\Omega = \{1, 2, 3, 4, 5, 6\}$ und für das einmalige Werfen zweier fairer Würfel gilt $\Omega = \{1, 2, 3, 4, 5, 6\} \times \{1, 2, 3, 4, 5, 6\}$.

Zufallsereignisse

Ein **Ereignis** E ist eine Teilmenge von Ω. Zum Beispiel ist $E = \{6\}$ das Ereignis, dass die Zahl 6 gewürfelt wird und $E = \{2, 4, 6\}$ das Ereignis, dass eine gerade Zahl fällt. Die Menge aller möglichen Ereignisse auf einer Menge von Elementarereignissen ist die **Ereignisalgebra** $\mathcal{E}$.

Wahrscheinlichkeiten

Die Wahrscheinlichkeit eines Ereignisses $E \in \mathcal{E}$ ist gegeben durch ein **Wahrscheinlichkeitsmaß** $P : \mathcal{E} \to [0, 1]$. Zufallsereignisse können durch **Mengenoperationen** kombiniert werden. So ist die Wahrscheilichkeit für das gemeinsame Eintreten von A und B gegeben durch die Vereinigung der Mengen $P(A \cap B)$ und die Wahrscheinlichkeit für das Eintreten von A oder B gegeben durch die Vereinigung $P(A \cup B)$. Das Ereigniss, dass eine durch 2 teilbare Zahl mit einem Würfel geworfen wird ist gegeben durch $A = \{2, 4, 6\}$ und das Ereigniss dass eine Zahl die durch 3 Teilbar geworfen durch $B = \{3, 6\}$. Dann ist das Ereigniss eine Zahl zu würfeln die durch 2 *oder* 3 teilbar ist die Vereinigung von A und B $A \cup B = \{2, 3, 4, 6\}$ und das Ereigniss eine Zahl zu würfeln die durch 2 *und* 3 teilbar ist der Schnitt von A und B $A \cap B = \{6\}$

Zwei Zufallsereignisse A und B sind voneinander **unabhängig** wenn gilt

$$P(A \cap B) = P(A) \cdot P(B) \tag{2.1}$$

Für Zufallsereignisse die nicht unabhängig sind können wir die **bedingte Wahrscheinlichket** berechnen. Die Wahrscheinlichkeit von A abhängig von B wird notiert mit $P(A \mid B)$ und ist definiert als:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{2.2}$$

Die Wahrscheinlichkeit von A lässt sich nur berechnen falls die Wahrscheinlichkeit von B **wohldefiniert** ist, also $P(B) > 0$.

Wahrscheinlichkeitsräume

Eine Menge an Elementarereignissen Ω, eine σ-Algebra auf der Menge Ω die auf Ω und ein Wahrscheinlichkeitsmaß $P : \mathcal{E} \to [0, 1]$ bilden zusammen einen **Wahrscheinlichkeitsraum** $(\Omega, \mathcal{E}, P)$. Im folgenden beschränken wir uns auf diskrete Wahrscheinlichkeitsräume, welche dadurch gekennzeichnet sind, dass Ω abzählbar endlich ist [**ElementareStochastic**].

Zufallsvariablen

Eine **Zufallsvariable** X ist eine Abbildung von Ω in eine beliebige Menge C [**ElementareStochastic**]. Sei zum Beispiel Ω die Menge aller möglichen Ergebnisse des Werfens zweier fairer Würfel. Dann ist die Augensumme der Würfel gegeben durch die Abbildung $X : \Omega \to [0, 1]$ mit $X(i, j) := i + j$ für alle $(i, j) = w \in \Omega$. Die Wahrscheinlichkeit, dass die Augensumme 3 ergibt lässt sich dann berechnen mit

$$P(X = 3) = P(\{(1, 2), (2, 1)\}) = P((1, 2)) + P((2, 1)) = \frac{1}{36} + \frac{1}{36} = \frac{1}{18}$$

Analog zu Zufallsereignissen sind Zufallsvariablen unabhängig wenn gilt

$$P(X_1 = x_1 \mid X_2 = x_2, \dots X_n = x_n) = P(X_1 = x_1) \cdot P(X_2 = x_2) \cdot \dots \cdot P(X_n = x_n) \tag{2.3}$$

Stochastische Prozesse

Ein Stochastischer Prozess beschreibt ein System welches sich zu einem gegebenen Zeitpunk t in einem von endlich vielen Zuständen $S = \{s_1, s_2, \dots, s_n\}$ befinden kann, wobei das Verhalten des Systems durch Zufallsereignisse bestimmt wird. Formaler ausgedrückt ist ein stochastischer Prozess eine Menge an Zufallsvariablen $\{X_t\}_{t \in T}$, indiziert durch T [**StochasticProcesses**]. Die Zufallsvariablen sind eine Abbildung in den Zustandsraum S [**StochastischeProzesse**].$X_t \to S = \{s_1, s_2, \dots, s_n\}$.

TODO: Würfel Beispiel für Stochastischen Prozess einfügen

2.2 Markovketten und Markovmodelle

Definition 1 (Markov-Eigenschaft) *Man sagt ein stochastischer Prozess $\{X_t\}_{t \in T}$ besitzt die **Markov-Eigenschaft**, wenn der Zustand des nächsten Zeitpunktes $t + 1$ einzig und allein vom Zustand im gegenwärtigen Zeitpunkt t abhängt.*

$$P(X_{t+1} = i_{t+1} \mid X_t = i_t, \dots, X_2 = i_2, X_1 = i_1) = P(X_{t+1} = i_{t+1} \mid X_t = i_t) \tag{2.4}$$

Für alle $t \in T$ und alle $i_k \in S$ unter der Vorraussetzung, dass alle bedingten Wahrscheinlichkeiten Wohldefiniert sind, also $P(X_t = i_t, \dots, X_2 = i_2, X_1 = i_1) > 0$.

Definition 2 (Markovkette) *Eine Markovkette ist gegeben durch einen stochastischen Prozess $\{X_t\}_{t \in T}$, welcher Werte aus einer Menge an Zuständen $S = \{s_1, s_2, \dots, s_n\}$ annehmen kann und die Markov-Eigneschaft erfüllt. Solch ein Markovprozess wird beschrieben durch zwei Variablen. Einen **Startwahrscheinlichkeitsvektor** π mit Länge n und eine **Transitionsmatrix** A mit Dimension $n \times n$, wobei n die Anzahl der Zustände S ist. π_i gibt die Wahrscheinlichkeit an, in Zustand s_i zu starten. Also die Wahrscheinlichkeit, dass sich der Prozess in Zeitpunkt $t = 0$ in Zustand s_i befindet.*

$$\pi_i = P(X_0 = s_i) \tag{2.5}$$

Die Transitionsmatrix A beschreibt die bedingten Wahrscheinlichkeiten, mit denen ein Zustandsübergang geschieht. $a_i(j)$ ist die Wahrscheinlichkeit dass sich der Prozess in Zeitpunkt $t + 1$ in Zustand s_j befindet, unter der Bedingung das er sich in Zeitpunk t in Zustand s_i befindet.

$$a_{ij} = P(X_{t+1} = s_j \mid X_t = s_i) \tag{2.6}$$

Eine Markovkette ist **zeithomogen** wenn die Transitionswahrscheinlichkeiten a_{ij} unabhängig von der Zeit t sind [**HmmIntroduction**].

$$a_{ij} = P(X_{t+1} = s_j \mid X_t = s_i) = P(X_t = s_j \mid X_{t-1} = s_i) \tag{2.7}$$

In dieser Arbeit werden ausschließlich zeithomogene Markovketten betrachtet.

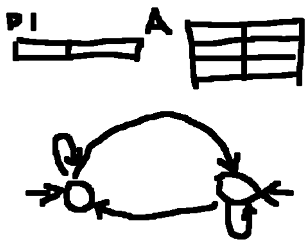
Das Ereignis in irgendeinem Zustand zu starten, so wie das Ereignis von einem gegebenen Zustand in irgendeinen anderen Zustand zu wechseln sind sichere Ereignisse. Somit gelten die Bedingungen, dass die Werte des Startwahrscheinlichkeitsvektors π und die Werte der ausgehenden Transitionen für jeden Zustand A_i in Summe jeweils 1 ergeben müssen. Diese Bedingung nennt man **Reihenstochastizität**

$$\sum_{i=0}^{N} \pi_i = 1 \qquad\qquad \sum_{j=0}^{N} a_{ij} = 1 \tag{2.8}$$

Als Beispiel für eine zeitdiskrete homogene Markovkette werden wir die durchschnittliche jährliche Temperatur modellieren. Seien die Zustände des Modells gegeben durch $S = \{H, K\}$, wobei H für heiß und K für kalt steht. Der Startvektor π und die Transitionsmatrix A seien gegeben durch

$$\pi = (0.7, 0.3) \qquad\qquad A = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix} \tag{2.9}$$

Wir können diese Markovkette anschaulich als gerichteten Transitionsgraphen representieren.



Mit unserem Temperaturmodell können wir nun zum Beispiel berechnen, was die Wahrscheinlichkeit ist die Temperaturabfolge $O = \{H, K, K, H, H\}$ zu beobachten.

$$\begin{aligned} P(O) &= P(O_1 = H) \cdot P(O_2 = K \mid O_1 = H) \cdot P(O_3 = K \mid O_2 = K) \\ &\cdot P(O_4 = H \mid O_3 = K) \cdot P(O_5 = H \mid O_4 = H) \\ &= \pi_1 \cdot a_{12} \cdot a_{22} \cdot a_{21} \cdot a_{11} \\ &= 0.7 \cdot 0.2 \cdot 0.6 \cdot 0.4 \cdot 0.8 = 0.02688 \end{aligned}$$

Eine Markovkette gibt uns also die möglichkeit probabilistische Systeme, in denen wir die Zustände und die Zuständsübergänge direkt beobachten können zu modellieren.

2.2.1 Hidden Markov Modelle (HMMs)

Bei einem Hidden Markov Modell ist die Observationssequenz nicht identisch zur Zustandssequenz, sondern die Observationssymbole sind das Ergebnis einer stochastischen Funktion der Zustände. Die Zustände selbst, so wie die Zustandsübergänge können wir *nicht beobachten* denn die Zustände sind „hidden". Ein HMM besteht also aus zwei stochastischen Prozessen. Ein versteckter Prozess, der die Transitionen in Abhängigkeit des vorherigen Zustandes bestimmt und ein zweiter Prozess, der die Observationssymbole in Abhängigkeit des gegenwärtigen Zustandes bestimmt. Formal definieren wir ein Hidden Markov Modell wie folgt.

Definition 3 (Hidden Markov Modell) *Ein Hidden Markov Modell ist definiert durch den Startwahrscheinlichkeitsvektor π und die Transitionsmatrix A einer Markovkette und einer zusätzlichen* ***Emissionsmatrix*** *$B = \{b_j(k)\}$ mit Dimension $N \times M$ wobei N die Anzahl der Zustände der Markovkette und M die Anzahl der möglichen Observationssymbole ist. Die Emissionsmatrix B gibt die Wahrscheinlichkeit eines Observationssymbols v_k unter der Bedingung, dass sich der Prozess im gegenwärtigen Zeitpunkt t in Zustand s_j befindet an.*

$$b_j(k) = P(O_t = v_k \mid q_t = S_j) \qquad 1 \leq j \leq N \qquad 1 \leq k \leq M \tag{2.10}$$

Wie ein solches HMM nun eine Observationssequenz O von Länge T erzeugen kann möchte ich durch folgendes Gedankenexperiment verdeutlichen. Stellen wir uns einen Raum vor, in dem N Urnen stehen. Die Urnen enthalten Kugel aus M verschiedenen Farben, wobei die Urnen verschiedene Verteilungen dieser Farben enthalten können. In dem Raum befindet sich eine Person welche nun insgesamt T Kugeln aus den Urnen ziehen und zurücklegen wird, wobei sie sich abhängig von einem Zufallsprozess von einer Urne zur nächsten bewegt. Wir können in diesen Raum nicht hineinsehen und somit nicht beobachten aus welcher Urne ein Kugel gezogen wurde. Wir kennen nur die Observationssequenz, also die Abfolge der Farben der gezogenen Kugeln. Um ethische Bedenken über dieses Gedankenexperiment zu minimieren sei angemerkt, dass der Raum klimatisiert ist und der Person Kakao und Kekse zur Verfügung stehen.

- Zunächst wird die initiale Urne anhand des Startwahrscheinlichkeitsvektor π ausgewählt.

- Dann zieht die Person eine Kugel aus der Urne, notiert deren Farbe als O_1 und legt die Kugel zurück.

- Danach wird eine neue Urne $q_2 = S_j$ mit Wahrscheinlichkeit $a_{ij} = P(q_2 = S_j \mid q_1 = S_1)$ gewählt. Aus dieser neuen Urne wird die nächste Kugel O_2 der Farbe v_k mit Wahrscheinlichkeit $b_j(k)$ gezogen.

Diese Prozedur wird so oft wiederholt bis der Zeitpunkt $t = T$ erreicht ist. Wir erhalten somit eine Observationssequenz von Farben $O = \{O_1, O_2, \ldots, O_T\}$. Abbildung **??** zeigt eine Visualisierung dieses Gedankenexperiments für $N = 3$

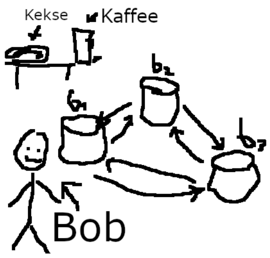


Figure 2.1: Urn Ball Model

2.3 Die drei klassichen Probleme

Für ein gegebenes Hidden Markov Model $\lambda = (A, B, \pi)$ und eine Observationssequenz O gibt es drei Probleme, welche von besonderem Interesse sind.

- **Problem 1** Was ist die Wahrscheinlichkeit, dass die Observationssequenz O von Modell λ erzeugt wurde $P(O \mid \lambda)$, und wie können wir diese effizient berechnen?
- **Problem 2** Welche Zustandssequenz $Q = q_1, q_2, \ldots, q_T$ hat die größte Wahrscheinlichkeit die Observationssequenz $O = O_1, O_2, \ldots, O_T$ zu erzeugen $\text{argmax}_Q\ P(Q \mid O, \lambda)$?
- **Problem 3** Wie können wir die Parameter des Modells λ anpassen um die Wahrscheilichkeit der Observationssequenz O zu maximieren? Anders formuliert suchen wir ein λ', so dass $P(O \mid \lambda') > P(O \mid \lambda)$

Auf diese Probleme wird jeweils in den folgenden drei Abschnitten eingegangen.

2.3.1 Berechnen der Wahrscheinlichkeit einer Observationssequenz

Bevor wir das Problem angehen wie man $P(O \mid \lambda)$ berechnet befassen wir uns zunächst damit wie man $P(O \mid Q, \lambda)$, also die Wahrscheinlichkeit von O wenn die Zustandssequenz $Q = q_1, q_2, \ldots, q_T$ gegeben ist, berechnen kann. Da wir in diesem Fall für jeden Zeitpunkt t das beobachtete Symbol O_t und den emittierenden Zustand q_t kennen ergibt sich die Berechnung von $P(O \mid Q, \lambda)$ aus der Definition von B.

$$P(O \mid Q, \lambda) = \prod_{t=1}^{T} b_{q_t}(O_t)$$

Ein intuitiver Ansatz $P(O \mid \lambda)$ zu Berechnen wäre nun alle Möglichen Zustandssequenzen Q aufzuzählen und die Wahrscheinlichkeiten $P(O \mid Q, \lambda)$ zu summieren.

$$P(O \mid \lambda) = \sum_{\text{alle Q}} P(O \mid Q, \lambda)$$

Es ist zwar möglich $P(O \mid \lambda)$ so zu berechnen, jedoch wächst die Anzahl der Möglichen Zustandsabfolgen Q exponentiell in Abhängigkeit zur Länge T von O. Insgesamt wären $2T \cdot N^T$ Berechnungen notwendig, was selbst bei kleinen Werten für N und T einen untragbaren Rechenaufwand darstellt. Für N=5 und T=100 wären es $2 \cdot 100 \cdot 5^{100} \approx 10^{72}$ Berechnungen. Um diese Zahl in Relation zu stellen, 10^{72} ist mindestens 3 mal mehr als 1000 und 1000 ist schon ziemlich groß. Zum Glück gibt es aber eine effiziente Methode um $P(O \mid \lambda)$ zu berechnen. Die Forwärtsvariable.

Der Forwärtsalgorithmus

Sei α, die Forwärtsvariable folgendermaßen definiert

$$\alpha_t(i) = P(O_1 O_2 \dots O_t \wedge q_t = S_i \mid \lambda)$$

$\alpha_t(i)$ Beschreibt also die Wahrscheinlichkeit, die partielle Observationssequenz $O_1 O_2 \dots O_t$ zu beobachten und in Zeitpunkt t in in Zustand i zu sein. $\alpha_t(i)$ kann folgendermaßen induktiv berechnet werden

Für $t = 0$ lässt sich $alpha_t(i)$ aus π und B berechnen, da noch keine Transition stattgefunden hat.

$$\alpha_0(i) = \pi_i \cdot b_i(O_0)$$

Für $t > 0$ gilt

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \cdot a_{i,j}\right] \cdot b_j(O_{t+1})$$

Der Casus Knacksus, welcher eine effiziente Berechung von α ermöglicht ist die Markov-Eigenschaft. Diese besagt, dass der Zustand in Zeitpunkt $t + 1$ nur vom Zustand in Zeitpunkt t abhängt. Für jeden Zeitpunkt und jeden Zustand gibt es also genau N Zustände in denen sich der Prozess zuvor befunden haben kann. Da der Rechenaufwand für jeden Zeitpunkt gleich ist hängt die Komplexität von $alpha$ mit $N^2 \cdot T$ nur noch linear von T ab und nicht exponentiell wie in dem vorherigen Ansatz.

2.3.2 Der Viterbi Algorithmus

Um die wahrscheinlichste Zustandssequenz zu berechnen müssen wir zunächst eine weitere Variable einführen, die Rückwärtsvariable β.

Definition 4 (Rückwärtsvariable β) *Die Rückwärtsvariable beta funktioniert analog zu α und ist wie folgt definiert.*

$$\beta_t(i) = P(O_{t+1}, O_{t+2} \ldots O_T \mid q_t = S_i, \lambda)$$

Somit ist $\beta_t(i)$ die Wahrscheinlichkeit in Zeitpunkt t in Zustand S_i zu sein und außgehend von S_i die partielle Observationssequenz $O_{t+1}, O_{t+2} \ldots O_T$ zu beobachten.

Für *$t = T$ gilt*

$$\forall_i \ 1 \leq i \leq N \mid \beta_T(i) = 1$$

Für *$t < T$ gilt*

$$\beta_t(i) = \sum_{j=1}^{N} a_{i,j} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)$$

Mit den Variablen α und β gewappnet können wir nun eine weitere Variable definieren.

$$\gamma_t(i) = P(q_t = S_i \mid O, \lambda)$$

Die Variable γ beschreibt die Wahrscheinlichkeit beim Beobachten von O in Zeitpunkt t in Zustand i zu sein. $\gamma_t(i)$ wird berechnet durch

$$\gamma_t(i) = \frac{\alpha_t(i) \cdot \beta_t(i)}{P(O \mid \lambda)} = \frac{\alpha_t(i) \cdot \beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i) \cdot \beta_t(i)}$$

Der Viterbi Algorithmus ähnelt dem Forwärtsalgorithmus sehr stark. Der maßgebliche Unterschied liegt jedoch darin, dass der Viterbi Algorithmus das **Maximum** der vorherigen Pfadwahrscheinlichkeiten betrachtet, wohingegen der Forwärtsalgorithmus die **Summe** betrachtet. Um die Wahrscheinlichkeit einer Observationssequenz zu berechnen ist es egal welcher Pfad

- Wir müssen uns nur den Pfad angucken, welcher am Wahrscheinlichsten in einen Zustand führt

Sei $\delta_t(i)$ die maximale Wahrscheinlichkeit, aller zusammenhängenden Pfade welche die ersten t Observationen erklären und in Zustand s_i enden.

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1 q_2 \ldots q_{t-1} \wedge q_t = i \wedge O_1 O_2 \ldots O_t \mid \lambda)$$
$$\delta_{t+1}(i) = (\max_i \delta_t(i) \cdot a_{ij}) \cdot b_j(O_{t+1})$$

Damit wir die wahrscheinlichste Zustandssequenz ausgeben können müssen wir uns für jeden Zeitpunkt t und jeden Zustand s_i das Argument, j welches $\delta_{t-1}(j)aij$ maximiert merken. Dies geschieht mittels des Arrays $\psi_t(i)$. Der ganze Lachs kann nun wie folgt gebutter werden.

- 1) Initialisierung

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(O_1) && 1 \leq i \leq N \\ \psi_1(i) &= 0 \end{aligned}$$

- 2) Rekursion

$$\begin{aligned} \delta_t(j) &= \max_{1\leq i\leq N}(\sigma_{t-1}(i)a_{ij}) \cdot b_j(O_t) && 2 \leq t \leq T, 1 \leq j \leq N \\ \psi_t(j) &= \operatorname*{argmax}_{1\leq i\leq N}(\delta_{t-1}(i)a_{ij}) && 2 \leq t \leq T, 1 \leq j \leq N \end{aligned}$$

- 3) Terminierung

$$\begin{aligned} P^* &= \max_{1\leq i\leq N}[\delta_T(i)] \\ q_T^* &= \operatorname*{argmax}_{1\leq i\leq N}[\delta_T(i)] \end{aligned}$$

- 4) Zustandssequenzrückverfolgung

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad t = T-1, T-2, \ldots, 1$$

2.3.3 Baum-Welch Algorithmus

Die letzte Variable, welche wir für den Baum-Welch Algorithmus benötigen ist $\xi_t(i,j)$, die vorwärts-rückwarts-Variable. Diese gibt die Wahrscheinlichkeit an in Zeitpunkt t in Zustand S_i zu sein und im nächsten Zeitpunk $t+1$ in Zustand S_j zu sein.

$$\xi_t(i,j) = \frac{\alpha_t(i) \cdot a_{i,j} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)}{P(O \mid \lambda)} = \frac{\alpha_t(i) \cdot a_{i,j} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i) \cdot a_{i,j} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)} \tag{2.11}$$

Der Baum Welch Algorithmus ist ein erwartungsmaximierender Ansatz. Er basiert auf dem Prinzip, dass wir aus den Variablen ξ und γ Erwartungswerte für die Anzahl an Transitionsübergängen und Symbolemissionen berechnen können. Mit dem Wissen wie oft diese Ereignesse erwartungsgemäß eintreffen können wir die Parameter des Models $\lambda = \pi, B, A$ neu berechnen. Wir erinnern uns

$\gamma_t(i) =$ Wahrscheinlichkeit in Zeitpunkt t in Zustand S_i zu sein

$\xi_t(i,j) =$ Wahrscheinlichkeit in Zeitpunkt t in Zustand S_i und in $t+1$ in S_j zu sein

Wenn wir $gamma_t(i)$ über alle t exklusive $t = T$ summieren erhalten wir die zu erwartende Anzahl an ausgehenden Transitionen von S_i

$\sum_{t=1}^{T-1} \gamma_t(i) =$ Zu erwartende Anzahl der von S_i ausgehenden Transitionen

Summieren wir $gamma_t(i)$ über alle t inklusive $t = T$ erhalten wir die zu erwartende Anzahl an Aufenthalten in Zustand S_i

$\sum_{t=1}^{T} \gamma_t(i) =$ Zu erwartende Anzahl an Aufenthalten in S_i

Die zu erwartende Anzahl der Transitionen von S_i zu einem bestimmten Zustand S_j erhalten wir ähnlich, durch summieren von $\xi_t(i,j)$ über alle $0 < t < T$.

$\sum_{t=1}^{T-1} \xi_t(i,j) =$ Zu erwartende Anzahl an Transitionen von S_i nach S_j

Mit den Erwartungswerten für Anzahlen an Zustandsübergängen, Zustandsaufenthalten und Zustandsemissionen können wir neue Werte für die Parameter des HMM berechnen.

$$\pi_i = \gamma_1(i) b_{i,j} = \frac{\sum_{t=1 s.d. O_t = V_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} a_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

Eine Neuberechnung der Parameter kann beliebig oft durchgeführt werden. In der Praxis wird als Abbruchbedingung ein Mindestwert für die Verbesserung des Models festgelegt.

Underflow und Overfitting

Wir erinnern uns, dass die Wahrscheinlichkeit einer Observationssequenz berechnet wird durch die rekursive variable $\alpha_t(i)$.

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \cdot a_{i,j} \right] \cdot b_j(O_{t+1})$$

Man kann erkennen, das $\alpha_t(i)$ expenontiell kleiner wird in Abhängigkeit von T. Das stellt ein Problem für die Implementation dar, denn eine binäre Representation einer Fließkommazahl erlaubt keine arbiträre Präzision. Eine Fließkommazahl die kleiner ist als wir mit unserem System darstellen können führt zu einem sogenannten **Underflow**. Um dieses ungewünschte Verhalten zu verhindern werden die Forwärts- und Rückwärtsvariablen **skaliert**. Beim skalieren werden die einzelnen $\alpha_t(i)$ und $\beta_t(i)$ mit **Skalierungskoeffizienten** multipliziert so dass sie sich stets im darstellbaren Bereich des Systems befinden. Der Nachteil davon ist, dass wir nur noch den Logarithmus der Wahrscheinlichkeit einer Observationssequenz berechnen können aber das nimmt man gerne in Kauf.

Ein weiteres Problem ist, dass Emissionen und Transitionen welche nicht in den Trainingsobservationen vorkommen mit 0 belegt werden. Dadurch kann es vorkommen, dass das trainierte Modell Observationen welche nicht im Trainingsset enthalten sind nicht akzeptiert. Wir sprechen bei solch einer fehlenden Generalisierung von **Overfitting**. Um dem entgegenzuwirken kann man ein **Smoothing**. Die einfachste Form eines Smoothing-Verfahrens besteht darin einen konstanten Wert auf alle Parameter zu addieren und diese anschließend wieder zu normalisieren. Solch ein Verfahren nennt man Laplace Smoothing.

2.4 Optimierungsverfahren

Im vorherigen Abschnitt haben wir uns unter anderem damit befasst wie die Parameter eines HMMs mittels des Baum-Welch Algorithmus optimiert werden können. In diesem Kapitel treten wir einen Schritt zurück und beantworten die Fragen was es überhaupt bedeutet Dinge zu optimieren und welche verschiedenen Optimierungsansätze existieren.

Was ist Optimierung

Wir sprechen von einem **Optimierungsproblem** wenn wir optimale Parameter eines Systems bestimmen wollen. Als Optimale Parameter bezeichnen wir solche, die eine Problemspezifische **Zielfunktion** f minimieren (oder maximieren). Die Domäne der Zielfunktion nennt man einen **Suchraum** S. Der Wert der Zielfunktion für eine Kombination von Parametern aus dem Suchraum gibt die **Qualität** dieser Parameter an. Oft gibt es Einschränkungen für die Parameter, so dass nicht alle Werte des Suchraumes mögliche Lösungen sind. Den Raum aller möglichen Lösungen nennen wir **zulässige Region** [**MetaheuristicsEGT**].

Um diese Konzepte zu verdeutlichen betrachten wir das bekannte Problem des Handlungsreisenden (traveling salesman problem), in welchem es darum geht den Besuch mehrerer Städte optimal zu planen, so dass jede Stadt außer dem Starpunkt nur einmal besucht wird und der Endpunkt equivalent zu dem Startpunkt ist. Der Suchraum ist in diesem Fall gegeben durch alle möglichen Routen (Kombinationen der zu besuchenden Städte). Die Zielfunktion, welche wir minimieren wollen gibt die gesamte Länge einer Reiseroute an. In der zulässigen Region des Suchraumes befinden sich ausschließlich Routen in welchen jede Stadt, bis auf den Startpunkt, einmal besucht wird und der Endpunkt equivalent zum Startpunkt ist. Da Teleportation noch nicht erfunden wurde ist die zulässige Region zusätzlich eingeschränkt auf alle Routen in welchen nacheinander besuchte Städte durch eine Strecke verbunden sind.

Kategorien von Optimierungsverfahren

Eine Optimierungsmethode, welche stets die global optimale Lösung findet nennt man eine **exakte Methode**. Für viele Probleme gibt es jedoch keine exakten Algorithmen die eine Lösung in polynomieller Zeit finden. In solch einem Fall kann man zu einer **Heuristik** greifen. Eine Heuristik liefert eine Lösung die "gut genug" ist in "annehmbarer Zeit". Es handelt sich also um ein Verfahren, welches man umgangssprachlich als Faustregel bezeichnen würde. Eine Heuristik ist nicht das selbe wie ein **Approximationsalgorithmus**. Denn ein Approximationsalgorithmus garantiert eine untere Schranke für die Qualität einer gefundenen Lösung. Bei einer Heuristik verhält es sich ähnlich wie mit Privatkäufen über Kleinanzeigenportale: Es besteht keine Garantie. Die gefundenen Lösungen eines heuristischen Verfahrens können also beliebig schlecht sein. Heuristiken werden unterteilt in **spezifische Heuristiken** und **Metaheuristiken**. Spezifische Heuristiken sind, wie der Name bereits vermuten lässt zugeschnitten auf ein spezifisches Problem, wohingegen Metaheuristiken sehr allgemein sind und auf fast alle Optimierungsprobleme angewendet werden können [**MetaheuristicsEGT**]. Metaheuristiken kann man weiter unterteilen in

Die wahrscheinlich bekannteste Heuristik ist eine **lokale Suche**. Der Baum-Welch Algorithmus zum Beispiel ist eine lokale Suche. Genetische Algorithmen, mit welchen wir uns später im Detail beschäftigen zählen zu den bekanntesten und ältesten Metaheuristiken.

Figur **??** zeigt die zuvor beschriebene Unterteilung der Optimierungsverfahren. Es sei angemerkt, dass die unternommene Unterteilung keineswegs Anspruch auf Vollständigkeit erhebt, sondern primär zur Einordnung des Baum-Welch Algorithmus und des genetischen Algorithmus im Kontext der Optimierungsverfahren dient.

Vergleich von Optimierungsalgorithmen

Zum Vergleich von Optimierungsalgorithmen ist es wichtig im Hinterkopf zu behalten, dass kein universal guter Optimierungsalgorithmus existieren kann. Das No Free Lunch Theorem postuliert, das für jedes Paar von Algorithmen, unter Betrachtung aller möglichen Optimierungsprobleme, beide Algorithmen im Durchschnitt gleich gut sind [**NFL**]. Daraus folgt: wenn Algorithmus A für eine Menge von Problemen besser ist als Algorithmus B muss Algorithmus B für alle restlichen Probleme besser sein als Algorithmus A.

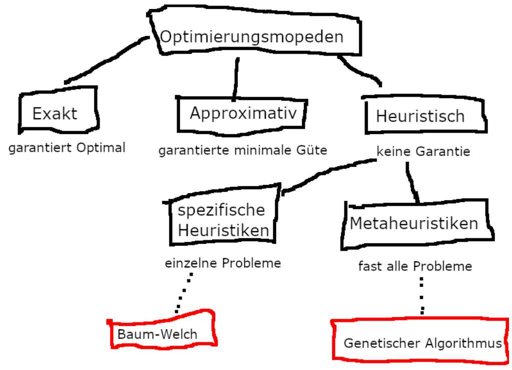


Figure 2.2: Unterteilung der Optimierungsverfahren

2.5 Metaheuristiken

Bei einer Metaheuristik handelt es sich nicht um einen bestimmten Algorithmus sondern vielmehr eine Ansammlung von Ideen, Konzepten und Operatoren, welche verwendet werden können um einen spezifischen heuristischen Algorithmus zu erstellen [**MetaheuristicsExposed**]. Es gibt also zum Beispiel nicht *den* genetischen Algorithmus sondern es existiert ein Genetischer Algorithmus "Bauplan" an welchem man sich orientieren kann.

Um eine Metaheuristik zu erstellen gilt es zwei konfliktierende Kriterien zu balancieren. Zum einen die **Diversifizierung** (exploration): das entdecken neuer Lösungen und zum anderen die **Intensivierung** (exploitation): das verbessern einer bekannten Lösung [**MetaheuristicsEGT**]. Reine Diversifizierung ist das selbe wie eine zufällige Suche und reine Intensivierung ist equivalent zu einer lokalen Suche.

Metaheuristiken sind oft inspiriert durch natürliche Prozesse. Bekannte Metaheuristiken aus dem Gebiet der Biologie sind **Genetische Algorithmen (GA)**, welche die Evolution einer Population durch natürliche Selektion nachahmen und **Ant Colony Optimization (ACO)** Algorithmen, welche der Pheromon-basierten Kommunikation von Ameisen innerhalb einer Kolonie nachempfunden sind. Zu den bekanntesten Beispielen aus der Physik zählt das **Simulated Annealing (SA)** Verfahren, welches angelehnt ist an den Abkühlungsprozess eines Metalls nach dem erhitzen [**metaheuristics**].

Hybride Heuristiken

Eine kombination mehrerer verschiedener Algorithmen zu einem neuen nennt man Hybridisierung. Das kombinieren von metaheuristischen Algorithmen ist ein aktives Forschungsfeld und es existiert mittlerweile eine beachtliche Anzahl hybrider Metaheuristiken.

Das kombinieren von Konzepten erlaubt es Wissenschaftlern außerhalb der Grenzen einer bestimmten Metaheuristik zu denken und ihrer Kreativität freien Lauf zu lassen [**MetaheuristicsSurvey**].

Es gibt verschiedene möglichkeiten der Hybridisierung. Wenn man verschiedene Heuristiken sequentiell hintereinander ausführt spricht man von einer **Relay Hybridization (RH)**. Eine Hybridisierung in der verschiedene Heuristiken kooperativ zusammenarbeiten nennt man eine **Teamwork Hybridization (TH)** [**MetaheuristicsEGT**].

Die Hauptmotivation hinter der hybridisierung ist es komplementäre Eigenschaften verschiedener Algorithmen auszunutzen [**MetaheuristicsSurvey**]. Populationsbasierte Metaheuristiken, wie zum Beispiel genetische Algorithmen sind gut im diversifizieren aber schlecht im intensivieren. Eine lokale Suche wiederum ist gut im intensivieren einer Lösung, bietet aber keine Diversifizierung. Es bietet sich also an Eine populationsbasierte Metaheuristik mit einer lokalen Suche zu kombinieren [**MetaheuristicsEGT**].

Parameter Tuning

Ein großer Nachteil vom Metaheuristiken ist, dass sie neue Parameter einführen, welche selbst optimiert werden müssen. Beispiele für solche Parameter sind die Mutationsrate eines Genetischen Algorithmus, die initiale Temperatur beim simulated annealing oder auch die Pheromonpersistenz eines Ant Colony Algorithmus. Eine Optimale Belegung dieser Parameter ist Problemabhängig. Daher gibt es keine Metaheuristik für welche universal optimale Parameter existieren. Man unterscheidet zwischen dem **Off-Line** und dem **On-Line** Parameter Tuning. In einem Off-Line Ansatz werden Werte für die Parameter vor der Ausführung des Algorithmus fixiert. Beim On-Line Parameter Tuning werden die Parameter während der Ausführung dynamisch angepasst [**MetaheuristicsEGT**].

Kritik an Metaheuristiken

Nach dem Ansturm an Metaheuristiken welchen wir in den letzten Jahren beobachten konnten hagelt es nun auch vereinzelt Kritik zur Verwendung dieser. Besonders hybride Metaheuristiken machen oft den Anschein, dass sich die Ersteller unzureichend bis gar nicht mit der Materie außeinandergesetzt haben. "Unfortunately, the used research methodology is often characterized by a rather ad hoc approach that consists in mixing different algorithmic components without any really serious attempts to identify the contribution of different components to the algorithms' performance" [**MetaheuristicsSurvey**]

So gibt es zum Beispiel viele Algorithmen welche eine hybridisierung zweier stochastischer populationsbasierter Metaheuristiken sind. Da die beiden Metaheuristiken sich nicht komplementieren ist das Rational hinter solch einer Hybridisierung schwer nachvollziehbar.

Vorallem die zunehmend absurder werdenden Metaphern hinter "neuen" Metaheuristiken ernten immer mehr Kritik. Ob Kakerlaken Befall [**MetaheuristicsExampleRoachInfestation**], Jazz-Musiker [**MetaheuristicsExampleHarmonySearch**], Schwarze Löcher [**MetaheuristicsExampleI**
oder auch intelligente Wassertropfen [**MetaheuristicsExampleIntelligentWaterDrops**]. Jedes erdenkliche Konzept kann in einen Optimierungsalgorithmus verwandelt werden. Oft stellt sich jedoch heraus dass das einzige was diese ,,neuen" Algorithmen zum Feld beitragen eine Umbenennung eines bereits etablierten Algorithmus ist [**NoNovelty**]. So ist zum Beispiel Harmony search, ein Suchalgorithmus der auf dem Prinzip von Jazz-Musikern funktioniert nichts weiteres als eine Umbenennung eines speziellen Falles des Genetischen Algorithmus [**HarmonySearch**]. Trotz mangelnder Innovation verzeichnet eine Suche nach ,,harmony search" auf Google Scholar laut Weyland im Jahre 2010 586 Einträge. Im Jahre 2023 ist diese Zahl auf stolze 57.500 Einträge gestiegen, wovon 7.840 Einträge nach 2022 erschienen. Die Flut solcher vermeintlich "neuen" Algorithmen ist sehr nachteilig für das Feld der Optimierung, denn die elaborierten Metaphern für bereits existierende Konzepte führen zu Verwirrung und tatsächlich innovative Ansätze werden übersehen [**MetaheuristicsExposed**].

2.6 Genetische Algorithmen

Evolutionäre Algorithmen sind stochastische populationsbasierte Metaheuristiken, welche dem Prozess der natürlichen Selektion nachempfunden sind. Die bekanntesten Paradigmen für Evolutionäre Algorithmen sind genetische Algorithmen (GA), evolution strategies (ES), evolutionary programming (EP), und genetic programming (GP) [**MetaheuristicsEGT**]. Im folgenden werde ich zunächst die Begrifflichkeiten, welche mit genetischen Algorithmen assoziiert sind erklären, dann einen Überblick über den Ablauf eines genetischen Algorithmus verschaffen und schließlich auf die einzelnen genetischen Operatoren eingehen und häufige Implementationen dieser aufzeigen.

genetische Algorithmen

- Wurden erstmals entwickelt durch J. Holland in den 1970 Jahren [missing quote EGT 384]

Ein genetischer Algorithmus arbeitet mit einer genetischen Representation der zu optimierenden Parameter [**TerminologiesAndOperators**]. Die genetische Representation werden wir im folgenden als **Chromosom (Genotyp)** und die originale Representation als **Phenotyp** bezeichnen. Ein Chromosom besteht aus mehreren **Genen** welche zu den

Parametern der Zielfunktion korrespondieren. Eine Menge an Chromosomen bezeichnet man als **Population**. Abbildung **??** verdeutlicht diesen Sachverhalt.

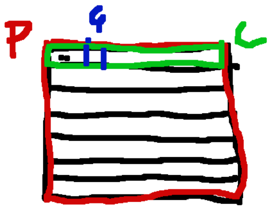


Figure 2.3: Gen, Chromosom und Population

Die **fitness** eines Chromosoms ist der Wert der Zielfunktion für den Phenotyp des Chromosoms. Also der Wert der Zielfunktion für die Parameter welche die Gene des Chromosoms representieren.

2.6.1 Ablauf eines genetischen Algorithmus

Der Ausgangspunkt eines genetischen Algorithmus ist eine Population an Chromosomen. Das erstellen dieser Population nennt man **Initialisierung**. Gewöhnlicherweise wird die initiale Population zufällig gewählt. Ein Iterationsschritt des klassischen genetischen Algorithmus besteht aus Vier Schritten [**GeneticAlgorithms**].

- Selektion: Aus der Population wird eine Menge an Eltern gewählt
- Crossover: Die Eltern werden zu einer Menge an Kindern Kindern kombiniert. Typischerweise erzeugen zwei Eltern-Chromosome ein Kind-Chromosom
- Mutation: Die Nachkommen werden mutiert, also zufällig verändert
- Evaluation: Jedem Kind wird ein fitness-Wert durch die fitness-Funktion zugewiesen
- Ersetzung: Die Alte Generation wird durch die neue ersetzt

Der Algorithmus terminiert wenn die vorher spezifizierte Anzahl an Generationen erreicht wurde. Alternative Terminierungskriterien sind zum Beispiel maximale Laufzeit des Algorithmus oder maximale Anzahl an Generationen ohne Verbesserung [**TerminologiesAndOperators**]. In Figur **??** wird der beschriebene Ablauf als Flussdiagramm dargestellt.

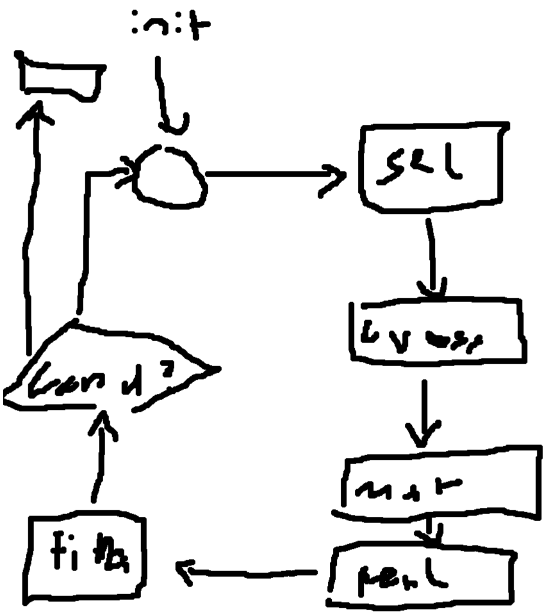


Figure 2.4: Flussdiagram eines genetischen Algorithmus

Selektionsoperator

Der Selektionsoperator wählt eine Menge von Eltern aus welchen Kinder für die nächste Generation erzeugt werden. Ganz nach der Evolutionstheorie sollen hier Chromosome mit einer höheren Fitness auch eine höhere Chance haben als Elternteil gewählt zu werden. Wie stark Chromosome mit einer höheren Fitness bevorzugt werden wird als **Selektionsdruck** bezeichnet. Selektionsstrategien können in zwei Klassen unterteilt werden, **proportionale Selektion** und **ordinale Selektion**. [**TerminologiesAndOperators**] Eine proportionale Selektion gewichtet Chromosome anhand ihrer Fitness. Ordinale Selektion gewichtet Chromosome anhand ihres Ranges. Bei einer proportionalen Selektion ist der Selektionsdruck hoch und es besteht das Risiko einer verfrühten Konvergenz. Denn wenn ein einzelnes Chromosom weitaus fitter, als der Rest der Population ist, wird dieses Chromosom einen proportionalen Selektionsprozess dominieren und somit die genetische Diversität der Population senken. Andererseits führt ein geringer Selektionsdruck zu langsamer konvergenz. [**TerminologiesAndOperators**] Die Auswahl des Selektionsoperators sollte also wohlüberdacht sein.

Zufällige Selektion funktioniert genau so wie der Name vermuten lässt. Die Eltern-Chromosome werden zufällig aus der Population gewählt so dass jedes Chromosom die gleiche Wahrscheinlichkeit hat als Elternteil gewählt zu werden.

Roulette Rad Selektion ist einer der traditionellen proportionalen Selektionsoperatoren. Stellen wir uns ein Roulette Rad vor, welches in N Segmente unterteilt ist, wobei N die Anzahl der Chromosome in einer Population ist. Die Länge eines Segmentes s_i ist proportional zu der normalisierten Fitness des korrespondierenden Chromosoms.

$$s_i = 2\pi \cdot \frac{fitness(i)}{\sum_{j=1}^{N} fitness(j)} \tag{2.12}$$

Nun wird das Roulette Rad gedreht und das Chromosom auf wessen Feld man landet wird in die Menge der Eltern aufgenommen. Diese Prozedur wird so oft wiederholt bis man die gewünschte Anzahl an Eltern gesammelt hat.

Rang Selektion ist eine Abwandlung der Roulette Rad Selektion. Die länge eines Segmentes ist jedoch nicht proportional zu der fitness sondern proportional zu dem Rang des Chromosoms.

$$s_i = 2\pi \cdot \frac{N - rank(i)}{\sum_{j=1}^{N} rank(j)} \tag{2.13}$$

Dadurch ist das Risiko einer verfrühten Konvergenz geringer als bei einer Roulette Rad Selektion.

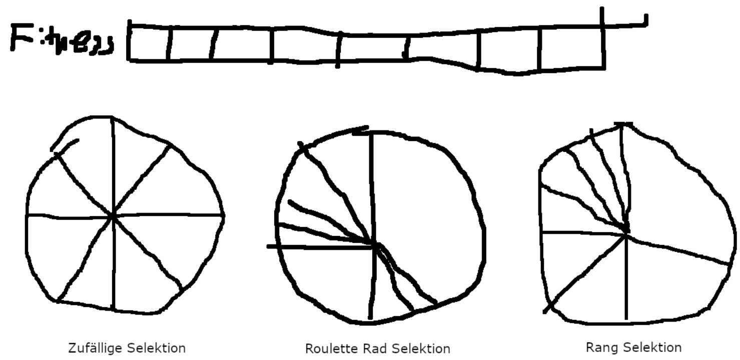


Figure 2.5: Verschiedene Selektionsoperatoren

Bei keinem der Selektionsoperator ist garantiert, dass die Chromosome mit der höchsten Fitness als Eltern ausgewählt werden. Darüber hinaus kann es vorkommen, dass Chromosome mit hoher Fitness durch Crossover und Mutation verschlechtert werden, so dass die maximale Fitness der folgenden Generation geringer als die vorherigen ist. Um eine Abnahme der maximalen Fitness zu verhindern wird **Elitismus** angewendet. Elitismus bedeutet dass einigie der besten Chromosome einer Population, die sogenannten Eliten in jedem Fall in die nächste Generation aufgenommen werden, ohne durch Crossover und Mutation verändert zu werden. Die Anzahl der Eliten ist variabel, um eine Abnahme der maximalen Fitness zu verhindern genügt es aber das Chromosom mit der höchsten Fitness zu übernehmen (Also ein Elitismus von 1).

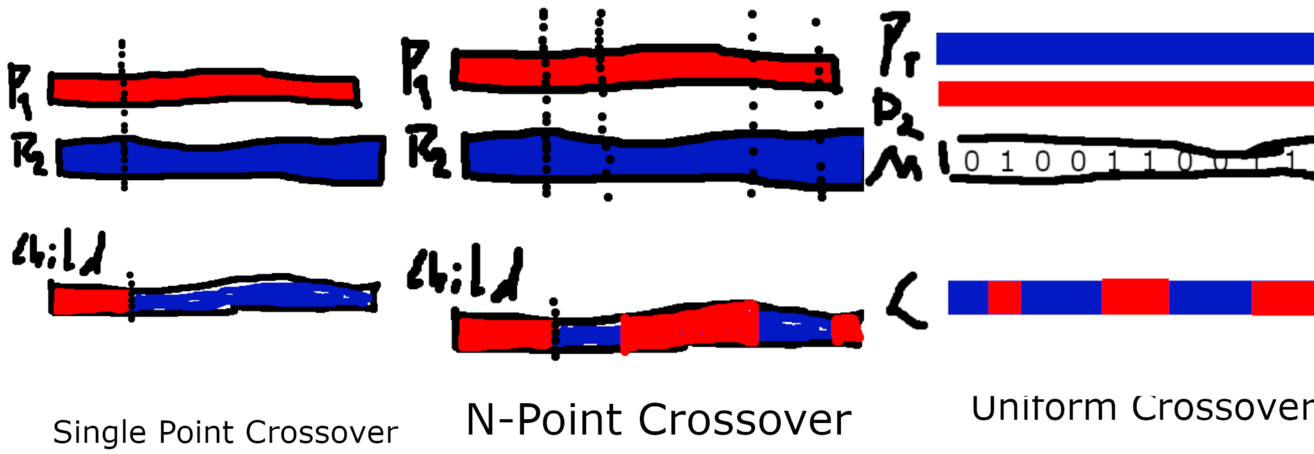
2.6.2 Crossoveroperator

Als Crossover bezeichnet man das Rekombinieren von typischerweise zwei Eltern-Chromosomen zu einem Kind-Chromosom. Es existieren auch Crossover-Operatoren, welche mehr als zwei Eltern akzeptieren oder mehr als ein Kind produzieren, diese werden wir jedoch nicht betrachten.

Beim **Single Point Crossover** wird ein Cutpoint entlang der länge der Eltern gewählt. Beide Eltern werden dann an diesem Cutpoint geschnitten und das Kind-Chromosom setzt sich zusammen aus der ersten Hälfte des ersten Elternteils und der zweiten Hälfte des zweiten Elternteils.

N-Point Crossover ist eine Generallisierung des Single Point Crossovers. Es werden n Crossover Points entlang der Länge der Eltern gewählt. Anhand dieser Crossover Points werden die Eltern in $n + 1$ Segmente unterteilt. Das Kind erhält alle Segmente mit geradem Index vom ersten Elternteil und alle Segmente mit ungeradem Index vom zweiten Elternteil. Im Allgemeinen führen mehr cutpoints jedoch zu einer geringeren Effizienz des genetischen Algorithmus. [**TerminologiesAndOperators**]

Bei einem **uniformen Crossover** wird eine Crossovermaske m mit gleicher Länge zu den Eltern erstellt. Das Kind erhält Gene der Eltern nach dieser Crossover Maske, wobei m_i angibt, von welchem Elternteil das i-te Gen bezogen wird. Für jedes Elternpaar wird eine neue Crossovermaske erstellt. Typischerweise gilt $P(m_i = 1) = P(m_1 = 0) = 0.5$. Die Wahrscheinlichkeit, dass ein Gen von einem Elternteil bezogen wird kann jedoch auch gewichtet werden anhand der Fitness oder Ränge, so dass $P(m_i = 0) = w$ und $P(m_i = 1) = 1 - w$



Mit der **Crossover-Wahrscheinlichkeit** kann man festlegen mit welcher Wahrscheinlichkeit ein Crossover zwischen zwei Eltern stattfindet. Für jedes Elternpaar wird eine zufällige Wahrscheinlichkeit generiert. Falls diese Wahrscheinlichkeit größer als die Crossover-Wahrscheinlichkeit ist findet kein Crossover statt und das Kind ist eine exakte Kopie eines Elternteils. So kann man festlegen wie viel genetische Information in der nächsten Generation erhalten bleibt.

2.6.3 Mutationsoperator

Das mehrfache Anwenden von Crossover und Selektion verringert unweigerlich die genetische Diversität der Population. Um dem entgegenzuwirken muss es einen Mechanismus geben, welcher neues genetisches Material hinzufügt. Dieser Mechanismus ist die Mutation, welche ein Chromosom zufällig verändert. Mutation garantiert, dass der Suchraum des genetischen Algorithmus **ergodisch** ist, was bedeutet das jeder Punkt im Suchraum von jedem anderen erreichbar ist [**TerminologiesAndOperators**]. Welche Mutationsoperatoren man verwendet hängt stark vom Problem und der Kodierung eines Chromosoms ab, da viele Mutationsoperatoren nur für bestimmte Arten von Genen anwendbar sind. Ein Bitflip Mutationsoperator zum Beispiel funktioniert logischerweise nur für eine binäre Kodierung. Für diese Arbeit interessieren uns nur Mutationsoperatoren die mit Fließkommazahlen arbeiten.

Die **Mutationsrate** bestimmt wie viele Gene eines Chromosoms durch die Mutation verändert werden und wie viele Unverändert bleiben. Bei einer Mutationsrate von 100% werden alle Gene eines Chromosoms verändert und der genetische Algorithmus ist equivalent zu einer zufälligen Suche. [**TerminologiesAndOperators**]. Für die Mutationsrate sollten im Allgemeinen kleine Werte gewählt werden (wie 0.01 oder 0,001). Eine gute Faustregel ist eine Mutationsrate von $\frac{1}{k}$, wobei k die Anzahl der Gene ist [**MetaheuristicsEGT**].

Ein **N-Point Random Mutation** Operator wählt zufällig N Gene aus die Mutiert werden sollen und ersetzt diese dann durch zufällige Werte.

Bei einer **Uniform Random Mutation** wird jedes Gen abhängig von der Mutationsrate mutiert.

TODO: Mutation aus ner alternativen Verteilung als Standardverteilung wählen.

Ersetzung

TODO: Ersetzungsstrategien

Parameter eines genetischen Algorithmus

Wie im Abschnitt [Abschnitt Label hier] erwähnt haben Metaheuristiken den Nachteil, dass sie neue Parameter einführen. Ein genetischer Algorithmus, wie er zuvor beschrieben wurde hat folgende Parameter.

1. Populationsgröße
2. Anzahl der Generationen (oder Abbruchbedingung)
3. Mutationsfunktion und Mutationsrate
4. Crossoverfunktion und Crossoverrate
5. Selektionsfunktion
6. Anzahl der Eliten
7. Ersetzungsstrategie

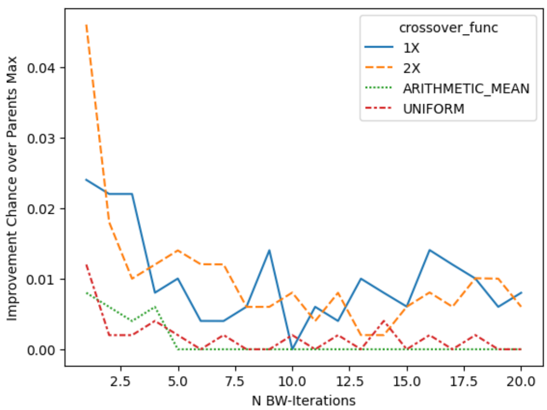
Die Belegung dieser Parameter ist keinesfalls trivial denn die Parameter beeinflussen die Konvergenzrate des Algorithmus maßgeblich. Die Populationsgröße zum Beispiel beeinflusst die globale Suchkapazität, ein größerer Wert ist hier von Vorteil. Jedoch benötigt eine große Populationi auch mehr Rechenleistung, Speicher und Zeit [**TerminologiesAndOperators**]. Die Mutationsrate beeinflusst ebenfalls die Konvergenzrate. Eine zu hohe Mutationsrate führt zu einer langsamen Konvergenz und einem ineffizienten Algorithmus. Eine zu niedrige Mutationsrate führt wiederum zu einer verfrühten Konvergenz und somit zu weniger optimierten Lösungen. Die Auswahl der genetischen Operatoren wirkt sich unter anderem auf die Rechenlast aus. Ein Single-Point-Crossover zum Beispiel generiert eine zufällige Zahl, und macht einen Vergleich. Die Anzahl der benötigten Zufallszahlen und Vergleiche bei einem Uniform-Crossover hingegen sind equivalent zu der Anzahl an Genen. Bei mehreren Hunderten von Genen macht sich dieser Unterschied durchaus bemerkbar.

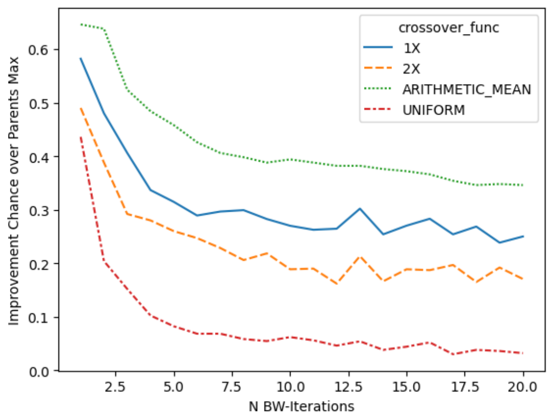
Chapter 3

Evaluation

Vergleich genetischer Operatoren

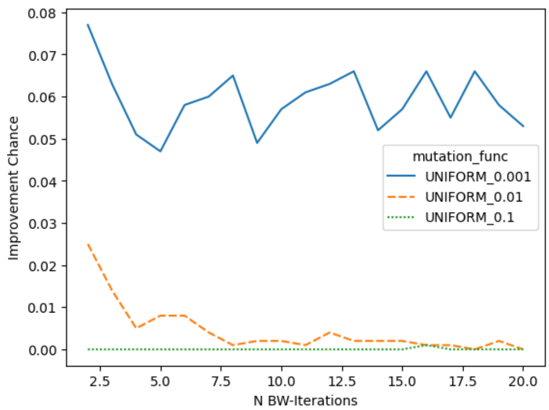
Zunächst wollen wir uns einen Überblick über die verschiedenen genetischen Operatoren verschaffen und diese vergleichen. Zu diesem Zweck wurden 1000 hidden Markov Modelle zufällig erstellt welche dann 20 Iterationen des Baum-Welch Algorithmus angewendet wurde. Der Observationssequenzen stammen hierbei aus dem Free Spoken Digit Dataset. Nach jedem Schritt des BW-Algorithmus wurden die verschiedenen Crossover und Mutationsoperatoren angewendet. So wurde die durchschnittliche Wahrscheinlichkeit ermittelt, dass ein gegebener Mutationsoperator ein Chromosom mit höherer Fitness erzeugt. Für die Crossoverfunktion wurde die durchschnittliche Wahrscheinlichkeit ermittelt, dass ein Kind eine höhere Fitness als seine Eltern hat.

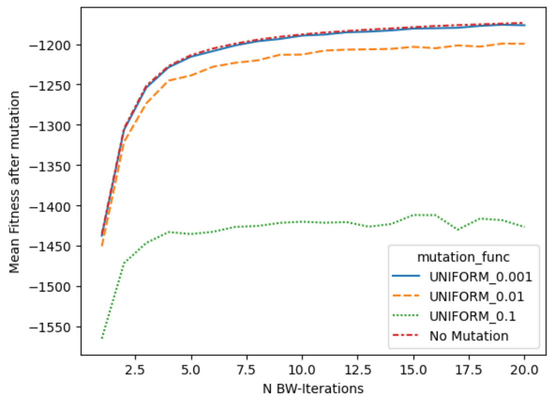




Wir können beobachten: je fitter die Eltern sind, desto geringer ist die Wahrscheinlichkeit, dass das Kind eine höhere Fitness als seine Eltern hat. Der Arithmetic Mean Crossover hat die höchste Wahrscheinlichkeit ein Kind zu erzeugen, welches eine höhere Fitness als ein mindestens ein Elternteil hat. Interessanterweise scheint der Arithmetic Mean Crossover jedoch nicht in der Lage zu sein Kinder zu Erzeugen, welche eine höhere Fitness als *beide* Elternteile haben. Der Uniforme Crossover schneidet am schlechtesten von allen ab und sollte nicht als Crossoveroperator eingesetzt werden. Der Single Point Crossover schlägt sich insgesamt am besten und wird im folgenden weiter betrachtet.

Eine Auswertung mit den selben Parametern wurde auch für die Mutationsoperatoren unternommen. Vergleicht wurde eine uniforme Mutation mit verschiedenen Werten für die Mutationschance. Es wurden die Werte 0.1, 0.01 und 0.001 als Mutationschance verglichen. Figur [MOPED] zeigt, Wahrscheinlichkeit, dass die Fitness eines Chromosoms nach der Mutation höher ist als zuvor abhängig von dem Grad der Optimierung (Anzahl der BW-Iterationen). Nur eine Mutationsrate von 0.001 ist verlässlig in der Lage Chromosome zu erzeugen, welche nach der Mutation eine höhere Fitness aufweisen. Figur [MOPED2] zeigt die durchschnittliche Fitness nach der Mutation im Vergleich zu keiner Mutation (nur Baum-Welch). Erneut beobachten wir, dass eine geringe Mutationsrate wünschenswert ist, da die Fitness der Population sonst enorm sinkt.





Natürlich ist solch ein isolierter Vergleich nur bedingt Aussagekräftig, da hier mögliche Interaktionseffekte der genetischen Operatoren nicht beachtet werden. Die Auswertung gibt uns jedoch eine Intuition über die Effektivität der Crossover- und Mutationsoperatoren. Bereits ab 5 BW-Iterationen, also schon ab einem geringen Maß an Optimierung, sinkt die Wahrscheinlichkeit ein Kind zu erzeugen, welches fitter als beide Eltern ist auf unter 2% für den single Point Crossover. Gleichzeitig liegt die Wahrscheinlichkeit mittels single Point Crossover ein Kind zu erzeugen, welches schlechter als beide Eltern ist nach 5 BW-Iterationen bei über 96%. Wir sehen also, dass der größte Teil unserer Rechenzeit dafür verwendet wird weniger optimierte Lösungen zu erzeugen.

Eine Käsestudie (Case Study)

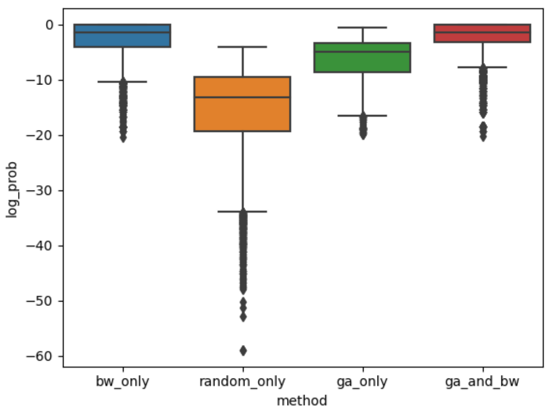
In diesem Abschnitt werden wir das GA-HMM Framework verwenden um einen existierenden Ansatz für die hybride Parameter-Estimation zu evaluieren. Der Ansatz welchen wir betrachten ist ,,Optimizing Hidden Markov Models with a Genetic Algorithm" [**LiteratureEvalGASlimane**] von M. Slimane et al aus dem Jahr 1996. Dieses

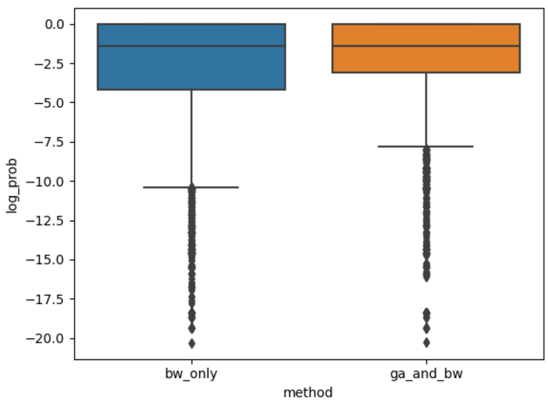
Paper wurde unter anderem ausgewählt, da es sich um einen der ältesten und meist zitierten Ansätze unter den hybriden Verfahren zur HMM-Parameter-Estimation handelt. Die Autoren verwenden einen genetischen Algorithmus um initiale Parameter für den Baum-Welch Algorithmus zu ermitteln. Es handelt sich also um einen Relay-Hybriden Ansatz. Es wurden 10 Observationssequenzen mit Längen zwischen 8 und 12 Observationen verwendet. Die Anzahl der Observationssymbole variiert zwischen 2 und 6. Als Strategie werden folgende Parameter verwendet

- Populationsgröße 60
- Elitismus 30
- Anzahl GA-Iterationen 200
- Selektionsstrategie Uniform zufällig
- Selektionsmenge: Die besten 30 Individuen
- Crossoveroperator: Single Point Crossover, jedoch nur am Ende von Reihen (1X)
- Mutationsoperator: Uniform Random Mutation
- Mutationschance: 0.01
- Rekombinationsstrategie: Generation wird bis auf Eliten komplett ersetzt
- Anzahl BW-Iterationen: 50

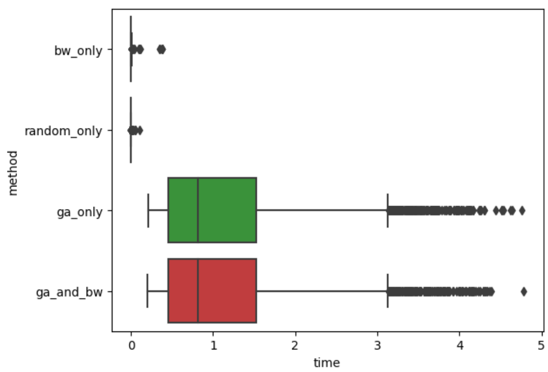
Parameterinitialisierungsstrategie besteht darin den ersten Parameter jeder Reihe beliebig im Interval $[0, 1]$ zu wählen und die folgenden parameter aus dem Intervall $[0, x]$, wobei x die Summe der vorherigen Parameter in der Reihe ist.

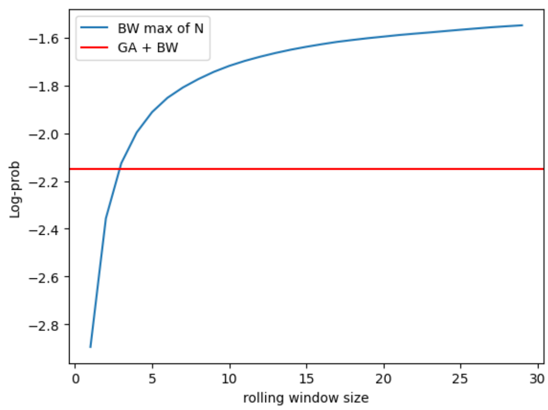
In ihren Auswertung bereichten Slimane et al, dass der hybride Ansatz im Durchschnitt bessere Hidden Markov Modelle erstellt. Ich konnte diese Ergebnisse verifizieren





Wir sehen, dass der genetische Algorithmus tatsächlich in der Lage ist im Durchschnitt bessere Ergebnisse zu finden als eine einfache Anwendung des BW-Algorithmus. Ein großer Nachteil des genetischen Algorithmus ist jedoch, dass er sehr Rechenintensiv ist. Grafik [SCHMOPED1] zeigt einen vergleich der durchschnittlichen Berechnungszeiten für eine Anwendung der jeweiligen Strategien. Die Durchführung des hybriden Algorithmus dauert durchschnittlich 0.997 Sekunden. Eine Anwendung von BW-Algorithmus alleine dauert durchschnittlich lediglich 0.002 Sekunden. Der hybride Algorithmus ist somit über 480x langsamer als der Baum-Welch Algorithmus alleine. Zu diesen Werten muss allerdings angeführt werden, dass der Baum-Welch Algorithmus vollständig optimiert ist und der genetische Teil des GA-HMM nur Teilweise optimiert. Andererseits ist ein genetischer Algorithmus einfach schwieriger zu optimieren, da man im Vergleich zum BW-Algorithmus enorm viele Parameter und Optionen verwalten muss. Ich bezweifle daher, dass eine weitaus performantere Implementation des genetischen Algorithmus möglich ist ohne dabei die Flexibilität und Lesbarkeit des Codes zu komprimieren. Dass der hybride genetische Algorithmus viel mehr Zeit benötigt ist per se kein Ausschluss-Kriterium. Auch wenn der hybride Algorithmus viel mehr Zeit benötigt, so ist eine maximale Rechendauer von unter 5 Sekunden wohl tollerierbar. Was uns interessieren sollte ist ob der hybride Algorithmus in der Lage ist Lösungen zu finden, welche ein reiner BW-Algorithmus in der selben Rechenzeit nicht finden kann. Um das zu überprüfen habe ich eine Anwendung des hybriden Algorithmus verglichen mit dem maximalen Wert aus N unabhängigen BW Anwendungen, wobei N von 1 bis 30 geht. Figur [SCHMOPED2] zeigt die Mittelwerte der Maximalen Fitness aus N BW Anwendungen. Der Mittelwert des hybriden Ansatzes ist ebenfalls als Referenz eingezeichnet. Wir sehen, dass bereits drei Anwendungen des BW-Algorithmus ausreichen um im Durchschnitt bessere Ergebnisse als der hybride Ansatz zu erzielen. Durch eine Erhöhung der Anzahl an BW Anwendungen kann die durchschnittliche Wahrscheinlichkeit sogar noch weiter erhöht werden. Durch mehrfache Anwendung des BW Algorithmus und wählen des wahrscheinlichsten Modells können wir also bessere Ergebnisse als durch den hybriden Ansatz für einen Bruchteil der Rechenzeit erhalten.





Eigenständigkeitserklärung

Hiermit versichere ich, dass ich diese Arbeit bzw. im Fall einer Gruppenarbeit den von mir entsprechend gekennzeichneten Anteil an der Arbeit selbständig verfasst habe. Ich habe keine unzulässige Hilfe Dritter in Anspruch genommen. Zudem habe ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle Ausführungen (insbesondere Zitate), die anderen Quellen wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht. Ich versichere, dass die von mir in elektronischer Form eingereichte Version dieser Arbeit mit den eingereichten gedruckten Exemplaren übereinstimmt. Mir ist bekannt, dass im Falle eines Täuschungsversuches die betreffende Leistung als mit "nicht ausreichend" (5,0) bewertet gilt. Zudem kann ein Täuschungsversuch als Ordnungswidrigkeit mit einer Geldbuße von bis zu 50.000 Euro geahndet werden. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuchs kann ich zudem exmatrikuliert werden. Mir ist bekannt, dass sich die Prüferin oder der Prüfer bzw. der Prüfungsausschuss zur Feststellung der Täuschung des Einsatzes einer entsprechenden Software oder sonstiger elektronischer Hilfsmittel bedienen kann.

Duisburg, 14. März 2023	
(Ort, Datum)	(Vorname Nachname)