

Bachelorarbeit

Fancy thesis template

Vorname Nachname
Matrikelnummer: 0123456
Angewandte Informatik (Bachelor)

UNIVERSITÄT
D U I S B U R G
E S S E N

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

6. 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 ist mittlerweile über 50 Jahre alt und in dieser Zeit wurden viele alternative Trainingsverfahren entwickelt, mit dem Ziel eine bessere globale Suchkapazität zu bieten. Viele dieser Algorithmen sind sogenannte Metaheuristiken welche oft den Baum-Welch Algorithmus hybridisieren. Ziel dieser Arbeit ist es zu evaluieren in wie weit solche (hybride) Metaheuristiken geeignet sind für die Parameter-Estimation eines HMM, wobei wir uns in der Auswertung auf genetische Algorithmen beschränken, welche zu den meistverwendeten 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

1	Introduction	1
2	Grundlagen	3
2.1	Grundbegriffe der Stochastik	3
2.1.1	Zufallsexperimente und Elementarereignisse	3
2.1.2	Zufallsereignisse	3
2.1.3	Wahrscheinlichkeiten	3
2.1.4	Wahrscheinlichkeitsräume	4
2.1.5	Zufallsvariablen	4
2.1.6	Stochastische Prozesse	4
2.2	Markovprozesse und Markov Modelle	5
2.2.1	Markovketten	5
2.2.2	Hidden Markov Modelle	6
2.3	Die drei klassischen Probleme	7
2.3.1	Berechnen der Wahrscheinlichkeit einer Observationssequenz	8
2.3.2	Der Viterbi Algorithmus	9
2.4	Baum-Welch Algorithmus	11
2.5	Optimierungsverfahren	13
2.5.1	Was ist Optimierung	13
2.5.2	Kategorien von Optimierungsverfahren	13
2.5.3	Metaheuristiken	14
2.5.4	Hybride Heuristiken	15
2.5.5	Parameter Tuning	15
2.5.6	Kritik an Metaheuristiken	16
2.6	Genetische Algorithmen	17
2.6.1	Ablauf eines genetischen Algorithmus	17
2.6.2	Crossoveroperator	20
2.6.3	Mutationsoperator	21
3	Konzept	25
3.1	Implementierung	26
4	Evaluation	31
5	Conclusion	33

Bibliography

35

Chapter 1

Introduction

Hidden Markov Modelle bietet die Möglichkeit komplexe Systeme zu modellieren. Sie wurden im frühen zwanzigsten Jahrhundert durch Andrey Markov eingeführt [15]. 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 sind Hidden Markov Modelle finden praktische Anwendung in einer Vielzahl an wissenschaftlichen Disziplinen. Hidden Markov Modelle sind beliebt, da sie unter anderem sehr flexibel und einfach zu implementieren sind [5] und mittlerweile aus den Gebieten der Spracherkennung [12] und Bioinformatik [25] nicht mehr wegzudenken. Eine der prominentesten Anwendungen von Hidden Markov Modellen ist der Google Pagerank Algorithmus, welcher die Relevanz von Website in Relation zu einer Suchanfrage ermittelt [14].

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 dominante 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

2.1.1 Zufallsexperimente und Elementarereignisse

Der Grundbaustein der Stochastik sind sogenannte **Zufallsexperimente**. Klassische Beispiele für Zufallsexperimente sind das Werfen eines fairen 6-seitigen Würfels 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 $\Omega = \{1, 2, 3, 4, 5, 6\} \times \{1, 2, 3, 4, 5, 6\}$.

2.1.2 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} . Sie entspricht der Potenzmenge der Elementarereignisse $\mathcal{E} = \mathcal{P}(\Omega)$.

2.1.3 Wahrscheinlichkeiten

Die Wahrscheinlichkeit eines Ereignisses $E \in \mathcal{E}$ ist gegeben durch ein **Wahrscheinlichkeitsmaß** $P : \mathcal{E} \rightarrow [0, 1]$. Zufallsereignisse können durch **Mengenoperationen** kombiniert werden. So ist die Wahrscheinlichkeit 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 $P(A \cup B)$. Zwei Zufallsereignisse A und B sind voneinander **unabhängig** wenn gilt

$$P(A \cap B) = P(A) \cdot P(B) \quad (2.1)$$

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

$$P(A | B) = \frac{P(A \cap B)}{P(B)} \quad (2.2)$$

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

2.1.4 Wahrscheinlichkeitsräume

Eine Menge an Elementarereignissen Ω , die auf Ω definierte Ereignisalgebra \mathcal{E} und das Wahrscheinlichkeitsmaß $P : \mathcal{E} \rightarrow [0, 1]$ bilden zusammen einen **Wahrscheinlichkeitsraum** (Ω, \mathcal{E}, P) . Im folgenden beschränken wir uns auf diskrete Wahrscheinlichkeitsräume, welche dadurch gekennzeichnet sind, dass Ω abzählbar endlich ist [4].

2.1.5 Zufallsvariablen

Eine **Zufallsvariable** X ist eine Abbildung von Ω in eine beliebige Menge C [4]. Sei zum Beispiel Ω die Menge aller möglichen Ergebnisse des Werfens zweier fairer Würfel. Dann ist die Augensumme der Würfel gegeben durch die Abbildung $X : \Omega \rightarrow [0, 1]$ 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

$$P(X_1 = x_1 | X_2 = x_2, \dots, X_n = x_n) = P(X_1 = x_1) \cdot P(X_2 = x_2) \cdots P(X_n = x_n) \quad (2.3)$$

2.1.6 Stochastische Prozesse

Ein Stochastischer Prozess beschreibt ein System welches sich zu einem gegebenen Zeitpunkt 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 ist eine Menge an Zufallsvariablen $\{X_t\}_{t \in T}$, indiziert durch T [26]. Die Zufallsvariablen sind eine Abbildung in den Zustandsraum S [9]. $X_t : \Omega \rightarrow S = \{s_1, s_2, \dots, s_n\}$

2.2 Markovprozesse und Markov Modelle

2.2.1 Markovketten

Eine Markovkette ist ein stochastischer Prozess in welchem der nächste Zustand einzig vom gegenwärtigen Zustand abhängt. Diese Eigenschaft ist bekannt als Markov-Eigenschaft [26].

Definition 1 (Markovkette) *Ein Stochastischer Prozess $\{X_t\}_{t \in T}$, welcher Werte aus einer Menge an Zuständen $S = \{s_1, s_2, \dots, s_n\}$ annehmen kann ist ein Markovprozess, falls gilt*

$$P(X_{n+1} = s_{n+1} \mid X_n = i_n, \dots, X_2 = i_2, X_1 = i_1) = P(X_{n+1} = i_{n+1} \mid X_n = i_n) \quad (2.4)$$

Für alle $n \in N_0$ und alle $i_k \in S$ unter der Voraussetzung, dass alle bedingten Wahrscheinlichkeiten Wohldefiniert sind, also $P(X_n = i_n, \dots, X_2 = i_2, X_1 = i_1) > 0$.

Für solch eine Markovkette definieren wir nun die zwei folgenden Variablen.

Definition 2 (Startwahrscheinlichkeitsvektor π) *π ist ein Vektor mit Länge $n = |S|$. 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.*

$$\pi_i = P(X_0 = s_i) \quad (2.5)$$

Definition 3 (Transitionsmatrix A) *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 dass er sich in Zeitpunkt t in Zustand s_i befindet.*

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

Eine Markovkette ist **zeithomogen** wenn die Transitionswahrscheinlichkeiten a_{ij} unabhängig von der Zeit t sind [10].

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

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

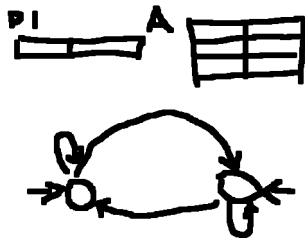
Werte der ausgehenden Transitionen für jeden Zustand A_i in Summe jeweils 1 ergeben müssen. Diese Bedingung nennt man **Reihenstochastizität**

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

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

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

Wir können diese Markovkette anschaulich als gerichteten Transitionsgraphen darstellen.



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

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

Eine Markovkette gibt uns also die Möglichkeit probabilistische Systeme, in denen die Zustände zu den Observationen korrespondieren zu modellieren.

2.2.2 Hidden Markov Modelle

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 können wir nicht beobachten denn sie 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. Dieser zweite Prozess wird beschrieben durch eine Emissionsmatrix B .

Definition 4 (Emissionsmatrix $B = \{b_j(k)\}$) 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.

$$b_j(k) = P(O_t = v_k \mid q_t = S_j) \quad 1 \leq j \leq N \quad 1 \leq k \leq M \quad (2.10)$$

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 und in welchem sich eine Person befindet. Wir können in diesen Raum nicht hineinsehen. Die Person wird nun insgesamt T Kugeln aus den Urnen ziehen und zurücklegen, wobei sie sich abhängig von einem Zufallsprozess von einer Urne zur nächsten bewegt. 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 $a_{ij} = P(q_2 = S_j \mid q_1 = S_i)$ 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 $O = \{O_1, O_2, \dots, O_T\}$. Abbildung 2.1 zeigt eine Visualisierung dieses Gedankenexperiments für $N = 3$

2.3 Die drei klassischen 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?

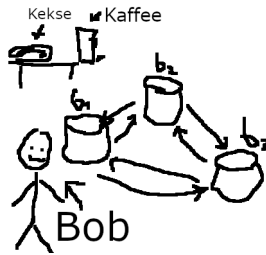


Figure 2.1: Urn Ball Model

- **Problem 2** Welche Zustandssequenz $Q = q_1, q_2, \dots, q_T$ hat die größte Wahrscheinlichkeit die Observationssequenz $O = O_1, O_2, \dots, O_T$ zu erzeugen $\arg\max_Q P(Q | O, \lambda)$?
- **Problem 3** Wie können wir die Parameter des Modells λ anpassen um die Wahrscheinlichkeit der Observationssequenz O zu maximieren? Anders formuliert suchen wir ein λ' , so dass $P(O | \lambda') > P(O | \lambda)$

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 $P(O | \lambda)$ berechnet befassen wir uns zunächst damit wie man $P(O | Q, \lambda)$, also die Wahrscheinlichkeit von O wenn die Zustandssequenz $Q = q_1, q_2, \dots, q_T$ 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 $P(O | Q, \lambda)$ aus der Definition von B .

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

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

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

Es ist zwar möglich $P(O | \lambda)$ 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 $P(O | \lambda)$ zu berechnen. Die Forwardvariable.

Der Forwardalgorithmus

Sei α , die Forwardvariable 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 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 Berechnung 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 befinden haben kann. Da der Rechenaufwand für jeden Zeitpunkt gleich ist hängt die Komplexität von α 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 5 (Rückwärtsvariable β) Die Rückwärtsvariable β funktioniert analog zu α und ist wie folgt definiert.

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

Somit ist $\beta_t(i)$ die Wahrscheinlichkeit in Zeitpunkt t in Zustand S_i zu sein und ausgehend von S_i die partielle Observationssequenz $O_{t+1}, O_{t+2} \dots O_T$ 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 Forwardalgorithmus sehr stark. Der maßgebliche Unterschied liegt jedoch darin, dass der Viterbi Algorithmus das **Maximum** der vorherigen Pfadwahrscheinlichkeiten betrachtet, wohingegen der Forwardalgorithmus 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.

$$\begin{aligned} \delta_t(i) &= \max_{q_1, q_2, \dots, q_{t-1}} P(q_1 q_2 \dots q_{t-1} \wedge q_t = i \wedge O_1 O_2 \dots O_t \mid \lambda) \\ \delta_{t+1}(i) &= (\max_j \delta_t(j) \cdot a_{ij}) \cdot b_i(O_{t+1}) \end{aligned}$$

Damit wir die wahrscheinlichste Zustandssequenz ausgeben können müssen wir uns für jeden Zeitpunkt t und jeden Zustand s_i das Argument, j welches $\delta_{t-1}(j) a_{ij}$ 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

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

- 3) Terminierung

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)]$$

$$q_T^* = \operatorname{argmax}_{1 \leq i \leq N} [\delta_T(i)]$$

- 4) Zustandssequenzrückverfolgung

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

Baum-Welch Verfahren

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

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

2.4 Baum-Welch Algorithmus

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 Ereignisse 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 $\gamma_{t+1}(i)$ 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 $\gamma_{t+1}(i)$ 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.

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

Eine Neuberechnung der Parameter kann beliebig oft durchgeführt werden. In der Praxis wird als Abbruchbedingung ein Mindestwert für die Verbesserung des Modells 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, dass $\alpha_t(i)$ exponentiell 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.

2.5 Optimierungsverfahren

Im vorherigen Abschnitt haben wir uns unter anderem damit befassen 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.

2.5.1 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** [23].

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 Startpunkt 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.

2.5.2 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 [23]. 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 2.2 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.

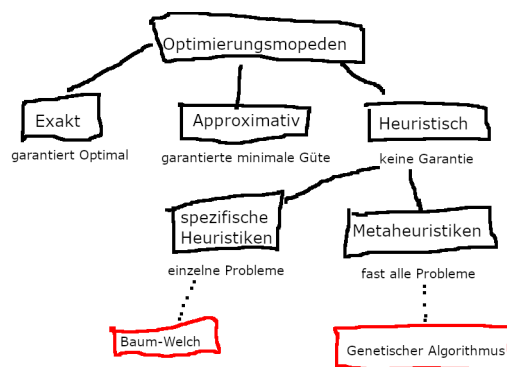


Figure 2.2: Unterteilung der Optimierungsverfahren

2.5.3 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 [21]. 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 [23].

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 [22].

2.5.4 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 [7].

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)** [23].

Die Hauptmotivation hinter der Hybridisierung ist es komplementäre Eigenschaften verschiedener Algorithmen auszunutzen [7]. 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 [23].

2.5.5 Parameter Tuning

Ein großer Nachteil von 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 [23].

2.5.6 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 auseinandergesetzt 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" [7]

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 Mikroflodermaus, Jazz-Musiker, Schwarze Löcher oder auch intelligente Wassertropfen. 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. [8] 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. [24] 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. [21]

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, dass für jedes Paar von Algorithmen, unter Betrachtung aller möglichen Optimierungsprobleme, beide Algorithmen im Durchschnitt gleich gut sind [6]. 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 .

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) [23]. 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 [19]. 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 2.3 verdeutlicht diesen Sachverhalt.

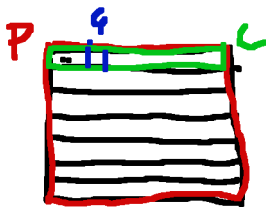


Figure 2.3: Gen, Chromosom und Population

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 [18].

- Selektion: Aus der Population wird eine Menge an Eltern gewählt
- Crossover: Die Eltern werden zu einer Menge an 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 [19]. In Figur 2.4 wird der beschriebene Ablauf als Flussdiagramm dargestellt.

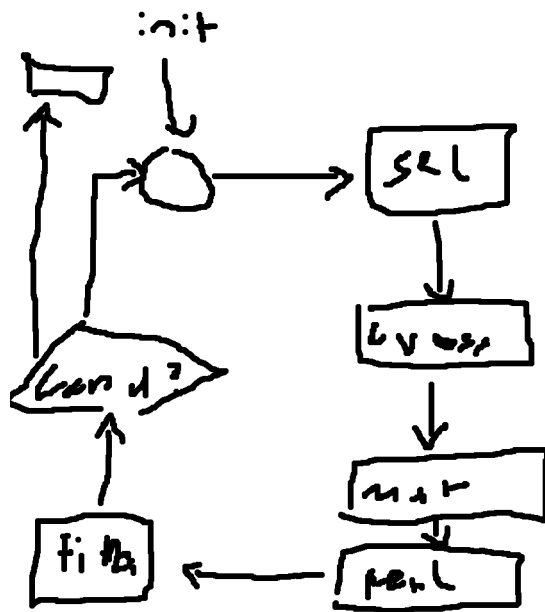


Figure 2.4: Flussdiagramm eines genetischen Algorithmus

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**. [19] 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. [19] 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.

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

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.

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

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

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

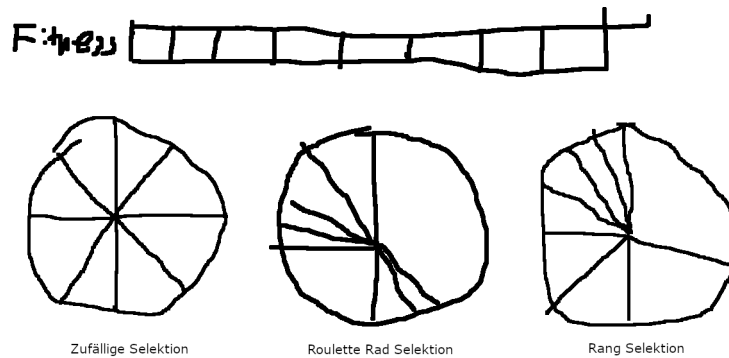


Figure 2.5: Verschiedene Selektionsoperatoren

bedeutet dass einige 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

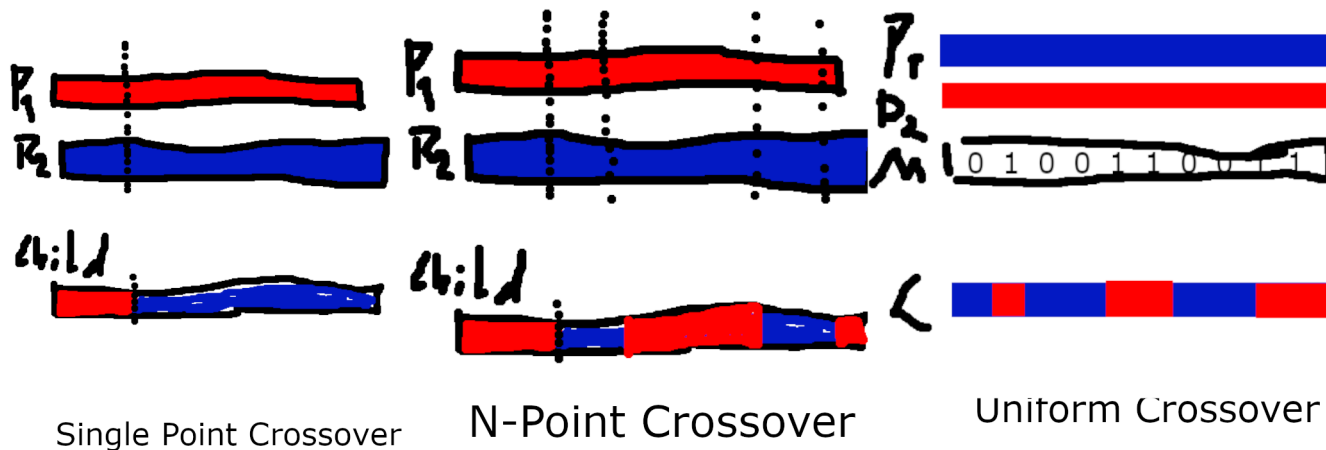
Als Crossover bezeichnet man das Rekombinieren von typischerweise zwei Eltern-Chromosomen zu einem Kind-Chromosom. Es existieren auch Crossover-Operatoren, welche mehr als zwei Eltern akzeptieren oder mehr als ein Kind produzieren, diese werden wir jedoch nicht betrachten.

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 Generalisierung 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. [19]

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 [19]. 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. [19]. 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 [23].

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 einer 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 Population auch mehr Rechenleistung, Speicher und Zeit [19]. 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

Konzept

Literaturübersicht

Es wurden bereits Zahlreiche Versuche unternommen das trainieren von HMMs durch Einbinden von Metaheuristiken zu verbessern. Darunter befinden sich Hybridisierungen mit klassischen Metaheuristiken wie Ant Colony Optimization [1] und Simulated Annealing [13] aber auch neuere metaphor-basierte Metaheuristiken wie modified gravitational search [2], Artificial Bee Colony [17] und chaos optimization [20] wurden auf das Problem der HMM Parameter optimierung angewendet. Es existieren sogar Ansätze, in welchen zwei Metaheuristiken mit dem Baum-Welch Algorithmus hybridisiert werden. Der GATSBW Algorithmus [16] zum Beispiel ist eine Kombination aus genetischen Algorithmus, Tabu-Suche und Baum-Welch Algorithmus. Neben dem Erlernen der Parameter wurden Metaheuristiken ebenfalls verwendet um eine passende Struktur (Anzahl der Zustände) zu finden [11].

Die Metaheuristischen Parameterestimationsverfahren lassen sich meinen Beobachtungen nach in drei Kategorien unterteilen.

- Rein metaheuristisch: Die Parameter werden nur durch eine Metaheuristik erlernt. Es kommt keine lokale Suche zum Einsatz.
- Erst Metaheuristik, dann BW: Die Metaheuristik wird verwendet um initiale Parameter für den Baum-Welch Algorithmus zu bestimmen (Relay-Hybrid).
- Metaheuristik und BW gleichzeitig: Eine Populationsbasierte Metaheuristik leitet die globale Suche. Lösungen werden mittels Baum-Welch intensiviert (Teamwork-Hybrid)

Aus den existierenden Ansätzen welche Hidden Markov Modelle mittels eines genetischen Algorithmus trainieren wurde für jede dieser drei Kategorien ein Ansatz herausgesucht.

3.1 Implementierung

Die ursprüngliche Idee zur Implementation eines GA-HMM Frameworks war es bereits existierende Python Libraries für HMMs und Genetische Algorithmen zu kombinieren. Da ich jedoch keine Libraries gefunden habe, welche meinen hohen Ansprüchen an Erweiterbarkeit genügen habe ich den Code von Grund auf konstruiert. Meine Erfahrung, des aus dieser Entscheidung resultierenden Schaffensprozesses, lässt sich adäquat durch folgendes Zitat beschreiben. "The basic idea behind the from scratch-oriented approach is the apparent simplicity of metaheuristic code. Programmers are tempted to develop themselves their codes. Therefore, they are faced with several problems: the development requires time and energy, and it is error prone and difficult to maintain and evolve" [23] Im folgenden werde die Funktionsweise des Frameworks beschreiben und darauf eingehen was bei einer Kombination von Hidden Markov Modellen mit einem genetischen Algorithmus zu beachten ist.

Representation

Wir erinnern uns, dass bei einem genetischen Algorithmus die Parameter in eine genetische Representation (Chromosom) überführt werden müssen. Die am häufigsten verwendete genetische Representation von HMM Parametern ist es die Matrizen der Parameter zu linearisieren und zu konkatenieren. Auch in dieser Arbeit werden wir solch eine Representation verwenden. Neben den parametern des Hidden Markov Models werden noch zusätzliche Informationen als Gene kodiert. Darunter sind Fitness und Rang so wie Anzahl der Zustände und Anzahl der Emissionssymbole. Die Anzahl der Zustände und Emissionsymbole werden als Gene kodiert, damit diese bei der Überführung von Chromosom-Representation zu HMM-Representation nicht als Parameter übergeben werden müssen. Die Kodierung der Fitness und des Rangs als Gene folgt einem ähnlichen Rational.

Die Fitness Funktion

Unser Ziel ist Parameter $\lambda = \pi, B, A$ zu finden welche die Wahrscheinlichkeit einer Menge an Observationssequenzen maximieren. Die Fitness Funktion eines Chromosoms ist daher die durchschnittliche Log-Wahrscheinlichkeit der Observationssequenzen für den Phänotyp des Chromosoms.

$$fitness(\lambda) = \frac{\sum_{\text{alle } O} \log[P(O | \lambda)]}{\text{anzahl } O}$$

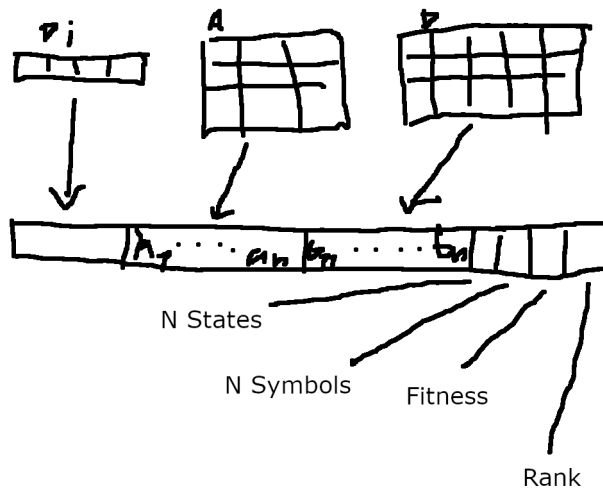


Figure 3.1: genetische Representation eines HMM

Genetische Operatoren

Die Genetischen Operatoren welche implementiert wurden sind

Crossover Operatoren

Mutations Operatoren

Um zu garantieren, dass die Struktur eines Hidden Markov Models nicht verändert wird gibt es verschiedene Kategorien. - Nur Emissionsmatrix mutieren - Maske erstellen und nur maskierte Werte Mutieren - Werte die Nicht verändert werden dürfen nicht in die Chromosom-Representation aufnehmen.

Ablauf des GA-HMM

Der GA-HMM erweitert den Ablauf eines klassischen genetischen Algorithmus um einige Schritte. Durch den Reproduktionsschritt können Chromosome erzeugt werden, welche unzulässige HMM-Parameter representieren. Zum Beispiel kann durch eine Mutation oder ein Crossover die Reihenstochastizität verletzt werden. Um solche unzulässigen Repräsentationen in den folgenden Schritten zu verhindern folgt nach dem Reproduktionsschritt ein **Normalisierungs-Schritt**. Solch eine Transformation unzulässiger

Lösung in zulässige Lösungen ist bekannt als **Reparatur Strategie** [23]. Des weiteren findet vor der Normalisierung ein optionaler **Smoothing-Schritt** statt um Overfitting entgegenzuwirken. Der Baum-Welch Algorithmus kann vor dem genetischen Algorithmus, danach und zwischendurch angewendet werden. Dadurch sind Relay-Hybridisierung, Teamwork-Hybridisierung und auch keine Hybridisierung innerhalb eines Frameworks durch Parameter für die einzelnen Baum-Welch Schritte realisierbar.

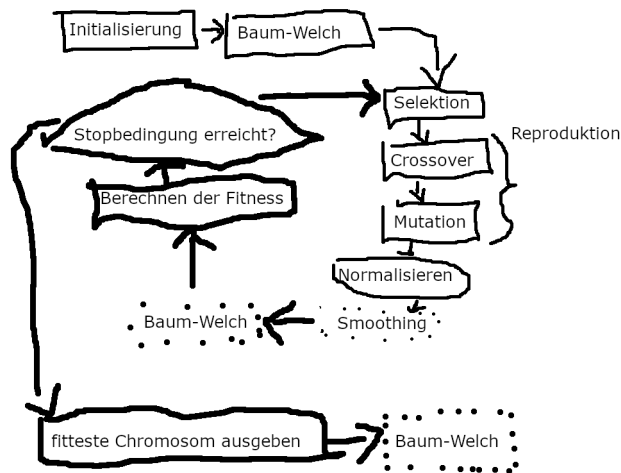


Figure 3.2: Ablauf des GA-HMM

Trainingsdaten

Zum Evaluieren der verschiedenen Trainingsverfahren wurden zwei Datensätze erstellt. Ein Datensatz wurde aus sprachlichen Äußerungen der Zahlen 0 bis 9 erstellt. Dazu wurden die Audiodateien zunächst in eine Vektorrepräsentation überführt. Sogenannte **Mel Frequency Cepstral Coefficients (MFCCs)**. Dabei handelt es sich um eine Darstellung des Frequenzspektrums welche dem menschlichen Hören nachempfunden ist. Diese MFCC-Vektoren wurden dann mit einem k-means Algorithmus quantisiert so dass man Observationssequenzen mit diskreten Symbolen erhält. Die Audiodateien wurden aus dem Free Spoken Digit Dataset (FSDD) entnommen. Ein zweiter Datensatz wurde aus Schwarz-Weiß Bildern von Gesichertern aus der ORL-Faces Datenbank generiert. Dazu wurden die Bilder zunächst linearisiert und anschließend wurden die Auflösung der Grau-Werte verringert um nicht allzuvielen Emissionsymbole zu erhalten.

Implementation des Baum-Welch

Der Baum-Welch Algorithmus wurde ebenfalls in python implementiert. Mittels des Numba JIT Compilers wird dieser Python-Code während der Laufzeit zu hochperformanten C-Code kompiliert. Um Kompilationszeit bei einem initialen Funktionsaufruf zu umgehen wird der kompilierte C-Code gecached. Die Korrektheit der Implementation wurde verifiziert indem ein HMM mit der gleiche Observationssequenz und gleichen initialen Parametern wie in Stamp [HIER STAMP REF EINFÜGEN] trainiert wurde. Die trainierten Parameter meiner Implementation weichen von den trainierten Parametern in Stamp um weniger als 0.0001 ab, was nach 100 Iterationen wohl im Bereich der durch Systemunterschiede erklärbaren Diskrepanz liegt.

Testing

Um die Korrektheit der Implementation des GA-HMM zu verifizieren wurde ein Domänen-spezifisches Test-Framework entwickelt. Die domänenspezifischen Methoden ermöglichen unter anderem Assert-Statements über Validität, Äquivalenz- und Ähnlichkeit der HMM und Chromosom Datentypen zu machen. Dadurch ist eine übersichtliche Validierung des Codes möglich.

Chapter 4

Evaluation

Vergleich genetischer Operatoren

Zunächst werden wir die verschiedenen vorgeschlagenen genetischen Operatoren miteinander vergleichen bezüglich der Kapazität eine gegebene Lösung zu verbessern. Dazu werden Crossover und Mutationsoperatoren auf Chromosome mit verschiedenen Optimierungsgrad angewendet. Wir können beobachten, dass sowohl Mutation als auch Crossover nur in den seltensten Fällen tatsächlich zu einer Verbesserung führen. Das zeigt, dass wir bei einem genetischen Algorithmus den größten Teil der Rechenleistung darauf verwenden Lösungen zu verschlechtern.

Performance Vergleich

In diesem Abschnitt vergleichen wir die Performance eines genetischen Algorithmus mit dem Baum-Welch Algorithmus und betrachten welche Schritte des GA-HMM am meisten Rechenzeit beanspruchen. Typischerweise ist bei einem genetischen Algorithmus die Fitnessfunktion am Rechenintensivsten [23]. Im Fall des GA-HMM ist die Fitness-Funktion jedoch sehr günstig. Es muss hier angemerkt werden, dass es sich nicht um einen fairen Vergleich handelt, da die Fitness-Funktion in optimierten C Code implementiert ist und der genetische Teil des GA-HMM nur teilweise optimiert ist. Ich bezweifle jedoch, dass man die Performance des genetischen Algorithmus stark verbessern kann ohne die Lesbarkeit oder Flexibilität des Codes zu komprimieren. Ein genetischer Algorithmus muss enorm viele Parameter festhalten und wurde daher als Objekt implementiert, welche zumindestens in Python sehr ineffizient sind. Der Baum-Welch Algorithmus hingegen kann als eine einzige Funktion implementiert werden, welche leicht optimiert werden kann.

Implementation verschiedener Ansätze

Ja sind alle kacke wer hätte gedacht?

Richard S. Barr schreibt, dass die Aussagekräftigste Visualisierung eine Darstellung der Qualität gegen Zeit ist. Perhaps the most popular and illuminating exhibits of heuristic performance is a graph of solution quality as a function of time. [3] Daher macht es mich stutzig warum gerade solch eine Visualisierung aus nicht in den Evaluationen der genetischen Algorithmen zu finden ist.

Chapter 5

Conclusion

Diskussion

Im Gebiet der metaheuristischen Optimierung gibt es meiner Beobachtung nach zwei Herangehensweisen. Einige verfolgen das Ziel zu verstehen warum Metaheuristiken funktionieren und wie diese bei Hybridisierungen miteinander interagieren. Andere wiederum haben augenscheinlich nur das Ziel für einen bestimmten Benchmark die höchste Punktzahl zu erhalten. Algorithmen mit Namen wie Hybrid simplified grey wolf and modified simple organism search (HSGWO-MSOS) oder auch hybrid least squares-support vector machine and artificial bee colony algorithm (ABC-LS-SVM) resultieren wohl aus letzterem Ansatz. Ich möchte keinen dieser Algorithmen verteufeln, es ist jedoch schwierig den Beitrag zum Stand der Wissenschaft, einer Aussage wie *"Algorithmus A hybridisiert mit Algorithmus B ist der beste für Problem X unter Parametern P"* zu bemessen. Abbildung 5.1 zeigt eine leicht überspitzte Darstellung dieses Sachverhaltes.

Mein hybrider kommunistische Ameisen und Chamäleon-Kolonie Algorithmus (KACK-Algorithmus) ist der beste im Vorraussagen, wann TK-Pizza bei ALDI im Angebot ist



Figure 5.1: Ein innovativer Algorithmus

Fazit

Ich denke die Ursprüngliche Frage, warum keine der innovativen Metaheuristiken zum trainieren von HMMs es geschafft auch nur annähernd die Prominenz des Baum-Welch Algorithmus zu erreichen sollte nun offensichtlich sein. Denn Metaheuristiken bringen eine Menge an Nachteilen mit sich.

- aufwendig zu implementieren und zu erweitern
- Rechenintensiv
- Sie führen Parameter ein, welche problemabhängig optimiert werden müssen.
- Parameter sind je nach Metapher schwierig zu interpretieren

Die Vorteile sind

- Besseres Ergebnis für ein spezifisches Problem

Man sollte also nur zu einer Metaheuristik greifen wenn man eine extrem Optimierte Lösung benötigt. "Only when very good solutions are needed which cannot be obtained by any complete method in a feasible time frame, the development of a hybrid metaheuristic is advised." [7]. Metaheuristiken sind kein Allheilmittel und sie sollten sparsam eingesetzt werden. Wie schon das alte Sprichwort lautet "Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel"

Ausblick

Eventuell ist es Sinnvoller metaheuristische Forschung auf Probleme zu konzentrieren für welche keine effizienten lokalen Optimierungsverfahren existieren. Das finden einer geeigneten Struktur für hidden Markov-Modelle zum Beispiel ist solch ein Problem.

Bibliography

- [1] Fatemeh Yassaee Meybodi Akram Emdadi Fatemeh Ahmadi Moughari and Changiz Eslahchi. *A novel algorithm for parameter estimation of Hidden Markov Model inspired by Ant Colony Optimization*. 2019.
- [2] Mohammadreza Ramzannezhad Ghaleh Ali Asghar Rahmani Hosseinabadi and Seyed Esmaeil Hashemi. *Application of Modified Gravitational Search Algorithm to Solve the Problem of Teaching Hidden Markov Model*. 2013.
- [3] Richard S. Barr. *Designing and reporting on computational experiments with heuristic methods*. 1995.
- [4] Ehrhard Behrends. *Elementare Stochastik*. 2012.
- [5] Ajay Kumar Bhavya Mor Sunita Garhwal. *A Systematic Review of Hidden Markov Models and Their Applications*. 2020.
- [6] William G. Macready David H. Wolpert. *No Free Lunch Theorems for Optimization*. 1997.
- [7] Nen paar dullis. *Paper mit nem tittel*. 2011.
- [8] *Grey Wolf Firefly and Bat Algorithms Three Widespread Algorithms that Do Not Contain Any Novelty*. 2020.
- [9] Dominik Wied Karsten Webel. *Stochastische Prozesse*. 2016.
- [10] Christian Kohlschein. *An introduction to Hidden Markov Models*.
- [11] Adam Prügel-Bennett und Anders Krogh Kyoung-Jae Won. *Training HMM structure with genetic algorithm for biological sequence analysis*. 2004.
- [12] Steve Young Mark Gales. *The Application of Hidden Markov Models in Speech Recognition*. 2008.
- [13] Douglas B. Paul. *Training of HMM Recognizers by Simulated Annealing*. 1985.
- [14] Arvind Lal Prerna Rai. *Google PageRank Algorithm - Markov Chain Model and Hidden Markov Model*. 2016.
- [15] Lawrence R. Rabiner. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. 1989.
- [16] Changhai Zhang Rengqin Yan and Ge Bai. *A Novel Genetic Algorithm Based on Tabu Search for HMM Optimization*. 2008.

- [17] K. Martin Sagayam and D. Jude Hemanth. *ABC algorithm based optimization of 1-D hidden Markov model for hand gesture recognition applications*. 2018.
- [18] Sivanandam. *Genetic Algorithms. In: Introduction to Genetic Algorithms*. Springer. 2008.
- [19] Sivanandam. *Terminologies and Operators of GA. In: Introduction to Genetic Algorithms*. Springer. 2008.
- [20] Rahati-Q Saeed Somayeh Cheshomi and Mohammah-R Akbarzadeh-T. *HMM training by a hybrid of Chaos Optimization and Baum-Welch algorithms for discrete speech recognition*. 2010.
- [21] Kenneth Sorensen. *Metaheuristics—the metaphor exposed*. 2012.
- [22] Springer. *Metaheuristics*. 2008.
- [23] El-Ghazali Talbi. *Metaheuristics - From design to implementation*. 1965.
- [24] Dennis Weyland. *A rigorous analysis vong dem harmony moped*. 2010.
- [25] Byung-Jun Yoon. *Hidden Markov Models and their Applications in Biological Sequence Analysis*. 2009.
- [26] Gordan Žitković. *Introduction to Stochastic Processes - Lecture Notes*. 2010.

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, 6. März 2023

(Ort, Datum)

(Vorname Nachname)