Universität Paderborn
Institut für Informatik
Warburger Strasse 100
33098 Paderborn

# Diplomarbeit

# Markowprozesse zur Modellierung bei Codekompression

Ulrich Hoppe

Betreuer: Prof. Dr. Johannes Blömer

Paderborn, 5. November 2003

# Inhaltsverzeichnis

**1 Einleitung** **1**

**2 Grundlagen** **3**
2.1 Datenkompression . . . . . 3
2.2 Die Sätze von Kraft und McMillan . . . . . 5
2.3 Entropie . . . . . 6
2.4 Kullback Leibler Distanz . . . . . 9
2.5 Codekompression . . . . . 12
2.6 Markowprozesse . . . . . 13

**3 Modelle** **19**
3.1 Bit-Modelle . . . . . 20
3.1.1 Markow-Kette erster Ordnung . . . . . 20
3.1.2 Zweite Phase - Kompressionsvorgang . . . . . 24
3.1.3 Markow-Kette zweiter Ordnung . . . . . 26
3.1.4 Markow-Kette dritter Ordnung . . . . . 27
3.1.5 Markow-Kette höherer Ordnung . . . . . 29
3.1.6 Testergebnisse . . . . . 31
3.2 2-Bit-Modelle . . . . . 33
3.2.1 Markow-Kette erster Ordnung . . . . . 33
3.2.2 Markow-Kette zweiter Ordnung . . . . . 36
3.2.3 Markow-Kette höherer Ordnung . . . . . 38
3.2.4 Testergebnisse . . . . . 40
3.3 Nibble-Modelle . . . . . 41
3.3.1 Markow-Kette erster Ordnung . . . . . 41
3.3.2 Markow-Kette zweiter Ordnung . . . . . 43
3.3.3 Testergebnisse . . . . . 45

**4 Reduktion der Modellgrösse** **47**
4.1 Konsolidierung von Wahrscheinlichkeitsverteilungen . . . . . 47
4.2 Knotenkonsolidierung im Nibble-Modell . . . . . 51
4.3 Algorithmen und Testergebnisse . . . . . 53
4.3.1 Erster Algorithmus . . . . . 53
4.3.2 Zweiter Algorithmus . . . . . 55
4.3.3 Optimaler Algorithmus . . . . . 58

**5 Fazit** **62**

**Literatur** **65**

**Index** **67**

# Tabellenverzeichnis


1 Testergebnisse der Bit-Modelle . . . . . . . . . . . . . . . . . . . . . 32
2 Testergebnisse der 2-Bit-Modelle . . . . . . . . . . . . . . . . . . . . 40
3 Testergebnisse der Nibble-Modelle . . . . . . . . . . . . . . . . . . . 46
4 Testergebnisse Modellreduktion - Erster Algorithmus . . . . . . . . . . 54
5 Testergebnisse Modellreduktion - Zweiter Algorithmus . . . . . . . . . 57
6 Testergebnisse Modellreduktion - Optimaler Algorithmus . . . . . . . 60

# Abbildungsverzeichnis


1 Bit-Modell - Markow-Kette erster Ordnung . . . . . . . . . . . . . . . 21
2 Bit-Modell - Markow-Graph erster Ordnung . . . . . . . . . . . . . . . 22
3 Beispiel eines Markow-Graphen erster Ordnung . . . . . . . . . . . . . 22
4 Positionsabhängige Modellierung . . . . . . . . . . . . . . . . . . . . 23
5 Bit-Modell - Markow-Kette zweiter Ordnung . . . . . . . . . . . . . . 26
6 Bit-Modell - Markow-Graph zweiter Ordnung . . . . . . . . . . . . . . 27
7 Bit-Modell - Markow-Kette dritter Ordnung . . . . . . . . . . . . . . . 28
8 Bit-Modell - Markow-Graph dritter Ordnung . . . . . . . . . . . . . . 29
9 Bit-Modell - Markow-Kette beliebiger Ordnung . . . . . . . . . . . . . 30
10 2-Bit-Modell - Markow-Kette erster Ordnung . . . . . . . . . . . . . . 34
11 2-Bit-Modell - Markow-Graph erster Ordnung . . . . . . . . . . . . . . 35
12 2-Bit-Modell - Markow-Kette zweiter Ordnung . . . . . . . . . . . . . 37
13 2-Bit-Modell - Ausschnitte eines Markow-Graph zweiter Ordnung . . 38
14 2-Bit-Modell - Markow-Kette der Ordnung k . . . . . . . . . . . . . . 39
15 Nibble-Modell - Markow-Kette erster Ordnung . . . . . . . . . . . . . 42
16 Nibble-Modell - Ausschnitt eines Markow-Graph erster Ordnung . . . 43
17 Nibble-Modell - Markow-Kette zweiter Ordnung . . . . . . . . . . . . 44
18 Nibble-Modell - Ausschnitt eines Markow-Graph zweiter Ordnung . . 45
19 Nibble-Modell - Zustände 00011101 und 10001100 . . . . . . . . . . . 52
20 Konsolidierung der Zustände 00011101 und 10001100 . . . . . . . . 52

# 1 Einleitung

In meiner Diplomarbeit wollen wir uns mit der Modellierung von Daten zwecks ihrer Kompression beschäftigen.

Es gibt prinzipiell zwei Möglichkeiten, eine gegebene Datenmenge zu komprimieren. Einerseits kann Redundanz in der Datenmenge ausgenutzt werden, um diese zu verkleinern. Andererseits kann man aber auch eine Häufigkeitsanalyse der in der Datenmenge auftretenden Symbole vornehmen, um diese effizienter durch Folgen anderer Symbole, etwa Folgen aus $\{0, 1\}^*$ zu codieren.

Zum besseren Verständnis, was Komprimierung von Daten aufbauend auf einer Häufigkeitsanalyse bedeutet, stellen wir uns folgendes Beispiel vor. Angenommen, wir wollen deutschsprachigen Text digital speichern. Dazu müssen wir jedem Buchstaben des Alphabets eine eindeutige binäre Darstellung zuordnen. Mit anderen Worten, jeder Buchstabe wird durch eine Kombination aus $\{0, 1\}^*$ repräsentiert.

Analysiert man nun einen gegebenen Text, so wird man in der Regel feststellen, dass der Buchstabe E viel häufiger vorkommt, als beispielsweise der Buchstabe X. Um den Text daher durch eine möglichst kurze Bitkonkatenation darzustellen, scheint es sinnvoll, den Buchstaben E durch eine sehr viel kürzere Bitfolge zu codieren, als den Buchstaben X.

Eine solche Analyse nennt man Modellierung oder auch Modellbildung für eine Datenkompression. Genau damit wollen wir uns in dieser Arbeit ausführlich beschäftigen. Im folgenden Kapitel werden wir zunächst eine einfache Modellierungsmöglichkeit zur Datenkompression kennenlernen. Diese basiert auf einer gegebenen Wahrscheinlichkeitsverteilung, die die Häufigkeit der Symbole einer gegebenen Datenmenge repräsentiert. Unter Verwendung dieser Wahrscheinlichkeitsverteilung kann ein geeignetes Verfahren die Datenmenge komprimieren.

Da diese Art der Modellierung aber für viele Anwendungen, in unserem Fall der Modellbildung für die Kompression ausführbarer Computerprogramme, nicht ausreichend ist, beschäftigen wir uns mit stochastischen Prozessen.

Unter Verwendung Markowscher Ketten lernen wir im dritten Kapitel bessere Modellierungsmöglichkeiten kennen. Hierbei berücksichtigen wir nicht nur die einzelnen Häufigkeiten der in einer Datenmenge auftretenden Symbole, sondern auch die Häufigkeit, wie oft ein Symbol direkt auf ein anderes Symbol folgt.

Bei der Analyse eines deutschen Textes würden wir also modellieren, wie oft beispielsweise das T dem S folgt. Häufigen Buchstabenkombinationen wie ST, SC oder EI würden also kürzere Bitkombinationen zugeordnet als Buchstabenkombinationen wie SZ oder SW.

Im dritten Kapitel wollen wir natürlich keinen deutschsprachigen Text, sondern die Binärdarstellung ausführbarer Programme mit Markowschen Ketten verschiedener Ordnung auf unterschiedliche Art und Weise analysieren. Die Modelle selbst werden wir anhand der zu erwartenden Kompression bewerten. Um dieses zu realisieren, betrachten wir Testergebnisse, die mit JAVA-Bytecode erstellt sind.

Diese Modelle besitzen allerdings eine etwas höhere Speicherkomplexität, so dass sie beispielsweise in eingebetteten Systemen nicht problemlos praktizierbar sind. Gerade in diesen Systemen, wo Speicherressourcen gering bemessen sind, darf ein Modell nicht zu gross werden. Um Speicherplatz zu sparen, werden dort ausführbare Programme komprimiert gespeichert, die allerdings zu ihrer Decodierung das Modell, sowie das Codierverfahren selbst benötigen. Man kann sich in diesem Fall also leicht überlegen, dass das Modell, mit dem die Kompression stattfindet, nicht zuviel Speicherplatz belegen sollte.

Zur Reduktion der Komplexität dieser Modelle, im Mittelpunkt steht hierbei das Modell, das in Abschnitt 3.3.2 vorgestellt wird, werden wir im vierten Kapitel verschiedene Algorithmen kennenlernen.

Besondere Beachtung bei einer Modellreduktion muss die zu erwartende Kompression finden, denn diese darf durch Veränderung, in diesem Fall durch Verkleinerung des Modells nicht zu schlecht werden. Eine gute Lösung wäre also ein Verfahren, das ein Modell auf eine praktikable Grösse reduziert, und gleichzeitig das Kompressionsverhalten nicht zu starkt negativ beeinflusst.

Nachdem wir schliesslich im vierten Kapitel diese Algorithmen betrachtet haben, endet diese Arbeit mit einem Fazit im fünften Kapitel.

# 2 Grundlagen

In diesem Kapitel werden grundlegende Begriffe aus dem Bereich der Datenkompression und der Stochastik erklärt. Diese bilden die Basis für den weiteren Verlauf meiner Arbeit. Unterstützend hierzu liefert Fachliteratur wie beispielsweise „Introduction to Data Compression“ [Say00], von Khalid Sayood oder „Elements of Information Theory“ [CT91], von Thomas M. Cover und Joy A. Thomas, entsprechende Informationen.

## 2.1 Datenkompression

Eine bekannte Technik zur Datenkompression ist die sogenannte Quellcodierung. Unter einer Quelle versteht man einen Strom, oder auch eine Konkatenation, aus einem endlichen Alphabet $A = \{a_1, \ldots, a_n\}$. Die einzelnen $a_i$ werden auch Worte genannt. Für diese Arbeit wollen wir uns auf binäre Worte beschränken. Es gilt also:

$$a_i \in \{0, 1\}^* \ \ \forall \ \ i \in \{1, \ldots, n\}.$$

Zu den Worten sind Wahrscheinlichkeiten gegeben, die die Häufigkeit der einzelnen Worte repräsentieren. Es existiert also zu jedem $a_i$ eine Wahrscheinlichkeit $p_i$. Damit wir eine Wahrscheinlichkeitsverteilung erhalten, gilt zudem

$$\sum_{i=1}^{n} p_i = 1.$$

Eine Quelle dieser Art nennt man *erinnerungslose Quelle*. Ziel der Quellcodierung ist es nun, die Worte aus $A$ möglichst effizient zu codieren. Effizient bedeutet hierbei, dass Worte mit grosser Wahrscheinlichkeit kurze Codierungen erhalten, und Worten mit geringer Wahrscheinlichkeit eine entsprechend längere Codierung zugeordnet wird.

**Definition 2.1.1** *Unter einer Codierung des Alphabets A versteht man eine injektive Abbildung*

$$C: \ A \ \rightarrow \ \{0, 1\}^* \quad , \quad a_i \ \mapsto \ C(a_i)$$

*Die Bilder $C(a_i)$ wollen wir mit $c_i$ bezeichnen. Diese heissen Codeworte. Die Abbildung C nennen wir einen Code.*

Unter einer *Nachricht* verstehen wir nun die Konkatenation von Worten aus $A^*$. Eine Codierung $C$ von $A$ kann auf Nachrichten $m = a_{i_1} \ldots a_{i_l} \in A^*$ erweitert werden, indem für $m = a_{i_1} \ldots a_{i_l}$ der Wert $C(m)$ definiert wird als $C(m) = C(a_{i_1}) \ldots C(a_{i_l})$.

Sei nun ein endlicher Bitstrom $C(m)$ durch einen Code $C$ erzeugt. Hierbei handelt es sich also um die codierte Nachricht $m \in A^*$. Möchte man von der codierten Nachricht auf die Nachricht selbst schliessen, ist man also an dem exakten $m \in A^*$ interessiert, so muss der Code $C$ *eindeutig entschlüsselbar* sein. Diese Eigenschaft führt uns direkt zur nächsten Definition.

**Definition 2.1.2** *Ein Code C heisst eindeutig entschlüsselbar, wenn die Abbildung*

$$C: \quad A^* \;\rightarrow\; \{\,0, 1\,\}^*$$

*injektiv ist.*

Anders ausgedrückt bedeutet diese Definition, dass ein Code $C$ eindeutig entschlüsselbar ist, falls jedes Element aus $\{\,0, 1\,\}^*$ Bild von höchstens einer Nachricht ist.

Um eine Nachricht korrekt decodieren zu können, reicht es also aus, dass der entsprechende Code eindeutig entschlüsselbar ist. Besitzt der Code zusätzlich die Eigenschaft, dass kein Codewort $c \in C$ existiert, das gleichzeitig Präfix eines anderen Codewortes $c' \in C$ ist, so handelt es sich um einen sogenannten Präfix-Code.

**Definition 2.1.3** *Ein eindeutig entschlüsselbarer Code C heisst Präfix-Code, falls keine Indizes $i, j$ mit $1 \leq i, j \leq n$, $i \neq j$, existieren, so dass $C(a_i) = c_i$ ein Präfix von $C(a_j) = c_j$ ist.*

Um die Unterschiede dieser Definitionen besser zu verstehen, betrachten wir folgendes Beispiel.

| $A$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ |
|---|---|---|---|---|
| $a_1$ | 0 | 0 | 0 | 0 |
| $a_2$ | 0 | 1 | 01 | 10 |
| $a_3$ | 1 | 00 | 011 | 110 |
| $a_4$ | 10 | 11 | 0111 | 1110 |

Die Spalten $C_i$ , $1 \leq i \leq 4$, entsprechen je einer Abbildung von $A \rightarrow \{\,0, 1\,\}^*$. Bei der Abbildung $C_1$ werden sowohl $a_1$ als auch $a_2$ auf 0 abgebildet. Das bedeutet jedoch, dass das Bild 0 zwei Urbilder in $A$ besitzt. Also ist die Abbildung $C_1$ nicht injektiv, und somit $C_1$ kein Code.

Die Abbildung $C_2$ ist injektiv. Demnach definiert $C_2$ zunächst einmal einen Code. Sei nun $C_2(m) = 00$. Betrachtet man die vier Codeworte von $C_2$, so stellt man fest, dass $C_2(m)$ Bild von $a_1a_1$ und auch Bild von $a_3$ sein kann. Der Code $C_2$ ist also kein eindeutig entschlüsselbarer Code.

$C_3$ hingegen ist ein eindeutig entschlüsselbarer Code. In diesem Fall kann eine codierte Nachricht $C_3(m)$ eindeutig zu $m$ decodiert werden, indem die Codierung von rechts nach links gelesen wird. Die 0 kennzeichnet jeweils den Beginn der Codierung eines Wortes aus $A$. Als kurzes Beispiel schauen wir uns die Codierung $C_3(m) = 0010000101$ an. Diese kann eindeutig zu $m = a_1a_2a_1a_1a_1a_2a_2$ decodiert werden.

Die Abbildung $C_4$ definiert sogar einen Präfix-Code. Man kann das sehr leicht überprüfen, indem man alle Präfixe von Codeworten mit allen Codeworten vergleicht.

Beim Code $C_4$ stellt sich dabei heraus, dass kein Codewort Präfix eines anderen Codewortes ist. Diese Eigenschaft entspricht genau der Definition 2.1.3.

Die Beispiele $C_3$ und $C_4$ verdeutlichen, dass eindeutig entschlüsselbare Codes und Präfix-Codes nicht ein und dasselbe sind.

Die Eigenschaften von Codes, eindeutig entschlüsselbar oder ein Präfix-Code zu sein, ermöglichen also die Decodierung von Nachrichten. Über die Kompressionseigenschaften der Codierung sagen sie jedoch nichts aus. Um auch über die Kompression einer Codierung eine Aussage treffen zu können, ist die *erwartete Länge eines Codes* $C$ wie folgt definiert.

**Definition 2.1.4** *Sei das Alphabet* $A = \{a_1, \ldots, a_n\}$ *mit der zugehörigen Wahrscheinlichkeitsverteilung* $P = \{p_1, \ldots, p_n\}$ *gegeben. Für einen Code* $C : A \to \{0, 1\}^*$ *bezeichnet*

$$E(C) := \sum_{i=1}^{n} p_i \, |C(a_i)|$$

*die erwartete Länge von* $C$*.* $E(C)$ *ist also der Erwartungswert der auf* $A$ *definierten Zufallsvariablen, die jedem Wort als Wert die Länge seiner Codierung zuweist.*

Im Rahmen der Datenkompression ist man natürlich an Codes interessiert, die eine gute Kompression erreichen, also deren erwartete Codelänge möglichst gering ist. Hierzu reicht es aus, sich auf Präfix-Codes zu beschränken.

**Definition 2.1.5** *Ein Code* $C$ *für eine Quelle* $Q$ *mit Alphabet* $A$ *und Wahrscheinlichkeitsverteilung* $P$ *heisst kompakt für diese Quelle, falls* $C$ *ein Präfix-Code mit minimaler erwarteter Länge für die Quelle ist.*

## 2.2 Die Sätze von Kraft und McMillan

Mit dem Wissen aus Abschnitt 2.1 können wir verstehen, dass es das Ziel der Datenkompression ist, kompakte Codes zu konstruieren. Hierbei kann man sich allerdings auf Präfix-Codes einschränken, denn zu jedem eindeutig entschlüsselbaren Code mit gegebenen Codierungslängen $l_1, \ldots, l_n$ für die Worte $a_1, \ldots, a_n \in A$, kann man einen Präfix-Code mit denselben Codierungslängen finden. Diese Aussage ist der Inhalt der folgenden Sätze.

**Satz 2.2.1 (Kraft)** *Ein Präfix-Code* $C$ *für das Alphabet* $A = \{a_1, \ldots, a_n\}$ *mit Codierungslängen* $|C(a_i)| = l_i\,, i = 1, \ldots, n\,,$ *existiert genau dann, wenn*

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1.$$

**Satz 2.2.2 (McMillan)** *Ein eindeutig entschlüsselbarer Code C für das Alphabet* $A = \{a_1, \ldots, a_n\}$ *mit Codierungslängen* $|C(a_i)| = l_i$ , $i = 1, \ldots, n$*, existiert genau dann, wenn*

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1.$$

Auf die Beweise wollen wir an dieser Stelle verzichten. Für Interessierte sind sie beispielsweise im Buch „Introduction to Data Compression“ [Say00], von Khalid Sayood nachzulesen. Aus den beiden Sätzen erhält man das folgende Korollar.

**Korollar 2.2.1** *Ein Präfix-Code für* $A = \{a_1, \ldots, a_n\}$ *mit Codierungslängen* $|C(a_i)| = l_i$ , $i = 1, \ldots, n$ *, existiert genau dann, wenn ein eindeutig entschlüsselbarer Code für A mit diesen Codierungslängen existiert.*

Zur Konstruktion von kompakten Codes reicht es also aus, sich auf Präfix-Codes zu beschränken.

Ein weiterer Vorteil der Präfix-Codes gegenüber eindeutig entschlüsselbaren Codes ist, dass sie häufig besser zu codieren und zu decodieren sind.

Somit kommen wir wieder zur Kompression, also zur erwarteten Länge eines Codes. Im folgenden Abschnitt werden wir uns eingehend damit beschäftigen.

## 2.3 Entropie

Um über die erwartete Länge eines kompakten Codes Aussagen treffen zu können, benötigen wir zunächst einmal den Begriff der *Entropie*.

**Definition 2.3.1** *Sei* $\{p_1, \ldots, p_n\}$ *eine endliche Wahrscheinlichkeitsverteilung, d. h.,* $0 \leq p_i \leq 1, i = 1, \ldots, n$ *, und* $\sum_{i=1}^{n} p_i = 1$*. Die Entropie dieser Wahrscheinlichkeitsverteilung ist definiert als*

$$H(p_1, \ldots, p_n) := -\sum_{i=1}^{n} p_i \log_2 p_i$$

Die Entropie ist ein Mass für die Zufälligkeit einer Wahrscheinlichkeitsverteilung.

**Beispiel:** Sei $P = \{p_1, p_2\}$ eine Wahrscheinlichkeitsverteilung mit $p_1 = p_2 = 1/2$. Die $p_i$ sind also gleich verteilt. Für die Entropie von $P$ ergibt sich dann

$$\begin{aligned} H(1/2, 1/2) &= -(1/2 \cdot \log_2(1/2) + 1/2 \cdot \log_2(1/2)) \\ &= -((-1/2) + (-1/2)) \\ &= 1 \end{aligned}$$

Für $n = 2$ ist $H(p_1, p_2)$ also maximal, wenn $p_1 = p_2 = 1/2$. Die Entropie ist hingegen 0, wenn die Wahrscheinlichkeitsverteilung keinerlei Zufall enthält, wenn also ein $i$, $1 \leq i \leq n$, existiert, für das $p_i = 1$ gilt.

**Hilfssatz 2.3.1** *1. Seien $\{p_1, \ldots, p_n\}$ , $\{q_1, \ldots, q_n\}$ zwei Wahrscheinlichkeitsverteilungen. Dann gilt*

$$H(p_1, \ldots, p_n) \leq -\sum_{i=1}^{n} p_i \log_2 q_i.$$

*2. Für jede Wahrscheinlichkeitsverteilung $\{p_1, \ldots, p_n\}$ gilt*

$$H(p_1, \ldots, p_n) \leq \log_2 n.$$

**Beweis:**

1. Sei ln der natürliche Logarithmus. Für alle $x > 0$ gilt $\ln x \leq x - 1$, wobei die Gleichheit nur bei $x = 1$ auftritt. Damit gilt dann auch $\ln(q_i/p_i) \leq (q_i/p_i) - 1$. Das liefert uns

$$\sum_{i=1}^{n} p_i \ln(q_i/p_i) \leq \sum_{i=1}^{n} q_i - \sum_{i=1}^{n} p_i = 0,$$

denn es gilt

$$\sum_{i=1}^{n} q_i = 1 \quad \text{und} \quad \sum_{i=1}^{n} p_i = 1.$$

Da $\ln(q_i/p_i) = \ln(q_i) - \ln(p_i)$ folgt

$$-\sum_{i=1}^{n} p_i \ln p_i \leq -\sum_{i=1}^{n} p_i \ln q_i.$$

Da $\ln x = \log e \cdot \log x$ folgt die Behauptung.

2. Die zweite Behauptung dieses Hilfssatzes folgt unmittelbar aus der Ersten mit $q_i = 1/n$ , $i = 1, \ldots, n$.

□

Mit diesem Hilfssatz sind wir nun in der Lage, einen Zusammenhang zwischen der Entropie der Wahrscheinlichkeitsverteilung einer Quelle $Q$ und der erwarteten Länge eines kompakten Codes für $Q$ herzuleiten.

**Satz 2.3.1** *Sei eine Quelle $Q$ mit Alphabet $A = \{a_1, \ldots, a_n\}$ und den zugehörigen Wahrscheinlichkeiten $\{p_1, \ldots, p_n\}$ gegeben. Für die erwartete Länge eines kompakten Codes $C$ für $Q$ gilt*

$$H(p_1, \ldots, p_n) \leq E(C) \leq H(p_1, \ldots, p_n) + 1.$$

**Beweis:** Zunächst wollen wir zeigen, dass $H(p_1, \ldots, p_n) \leq E(C)$ gilt. Hierzu sei $l_i = |C(a_i)|$ , $i = 1, \ldots, n$. Aus dem Satz von McMillan (Satz 2.2.2) folgt

$$p := \sum_{i=1}^{n} 2^{-l_i} \leq 1.$$

Wir setzen

$$q_i := \frac{2^{-l_i}}{p}, \quad i = 1, \ldots, n.$$

Auf diese Weise bilden die $q_i$ eine Wahrscheinlichkeitsverteilung. Nach dem Hilfssatz 2.3.1 gilt

$$H(p_1, \ldots, p_n) \leq -\sum_{i=1}^{n} p_i \log_2 q_i = \sum_{i=1}^{n} p_i l_i + \sum_{i=1}^{n} p_i \log_2 p = \sum_{i=1}^{n} p_i l_i + \log_2 p.$$

Nun ist $p \leq 1$ und somit $\log_2 p \leq 0$. Zudem ist

$$\sum_{i=1}^{n} p_i l_i = E(C).$$

Es folgt also

$$H(p_1, \ldots, p_n) \leq E(C).$$

Für den Beweis fehlt uns jetzt noch die Abschätzung $E(C) \leq H(p_1, \ldots, p_n) + 1$. Um dieses zu zeigen sei $l_i \in \mathbb{N}$ minimal mit der Eigenschaft $2^{-l_i} \leq p_i$. Daraus folgt

$$-\log_2 p_i \leq l_i \leq -\log_2 p_i + 1.$$

Da gilt

$$\sum_{i=1}^{n} 2^{-l_i} \leq \sum_{i=1}^{n} p_i = 1,$$

existiert nach dem Satz von Kraft (Satz 2.2.1) ein Präfix-Code $C'$ für $Q$ mit Codierungslängen $l_i$. Mit $l_i \leq -\log_2 p_i + 1$ gilt

$$E(C') = \sum_{i=1}^{n} p_i l_i \leq -\sum_{i=1}^{n} p_i \log_2 p_i + \sum_{i=1}^{n} p_i = H(p_1, \ldots, p_n) + 1.$$

□

Bei genauer Betrachtung dieses Beweises sieht man, dass die erwartete Länge des Codes $C$ genau dann der Entropie der Wahrscheinlichkeitsverteilung entspricht, wenn jedes der $p_i$ eine Zweierpotenz ist. Dies dürfte in der Praxis natürlich äusserst selten der Fall sein. Jedoch besteht durch diesen Satz die Möglichkeit, die erwartete Codelänge anhand der Entropie der Wahrscheinlichkeitsverteilung einzugrenzen.

Die erwartete Länge eines Codes $C$, für eine Quelle $Q$ mit Alphabet $A$ und einer gegebenen Wahrscheinlichkeitsverteilung $P$, kann also unter Verwendung der Entropie von $P$ abgeschätzt werden.

In diesem Abschnitt haben wir nun gesehen, dass die Entropie einer Wahrscheinlichkeitsverteilung ein geeignetes Hilfsmittel ist, um Aussagen über das Kompressionsverhalten einer Quelle treffen zu können.

Als nächstes wollen wir uns mit der Approximation einer Wahrscheinlichkeitsverteilung durch eine andere Wahrscheinlichkeitsverteilung beschäftigen. In diesem Zusammenhang werden wir im folgenden Abschnitt die sogenannte *Kullback Leibler Distanz* kennenlernen.

## 2.4 Kullback Leibler Distanz

Stellen wir uns vor, wir haben eine Quelle $Q$ mit Alphabet $A$ und gegebener Wahrscheinlichkeitsverteilung $P$. Aus den vorherigen Abschnitten wissen wir, dass die erwartete Codelänge $E(C)$ eines Codes $C$ der Entropie der Wahrscheinlichkeitsverteilung $P$ entspricht.

Benutzen wir jetzt aber eine andere Wahrscheinlichkeitsverteilung $R$, so können wir uns leicht überlegen, dass die Kompression nicht mehr der Entropie von $P$ entsprechen kann. Der resultierende Unterschied ist nämlich genau die Kullback Leibler Distanz der Wahrscheinlichkeitsverteilung $P$ zur Verteilung $R$ und Inhalt der nächsten Definition.

**Definition 2.4.1** *Seien zwei Wahrscheinlichkeitsverteilungen $P$ und $R$ gegeben durch $P = \{p_1, \ldots, p_n\}$ sowie $R = \{r_1, \ldots, r_n\}$. Die Kullback Leibler Distanz zwischen $P$ und $R$ ist dann definiert als*

$$D(P\|R) := \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{r_i}.$$

Im Rahmen dieser Definition sei zudem vereinbart, dass für $p_i = 0$ gilt $0 \log_2 \frac{0}{r_i} = 0$ und für $r_i = 0$ gilt $p_i \log_2 \frac{p_i}{0} \to \infty$.

Die Motivation dieser Definition begründet sich wie folgt:
Bei einer Codierung mit

$$\log_2 \frac{1}{p_i}$$

Bits ist die erwartete Codelänge gegeben durch

$$\sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}.$$

Codieren wir aber mit

$$\log_2 \frac{1}{r_i}$$

Bits, so ändert sich die erwartete Codelänge zu

$$\sum_{i=1}^{n} p_i \log_2 \frac{1}{r_i}.$$

Der Unterschied ist dann genau

$$\sum_{i=1}^{n} p_i \log_2 \frac{1}{r_i} - \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} = \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{r_i}.$$

Die Kullback Leibler Distanz zwischen zwei Wahrscheinlichkeitsverteilungen $P$ und $R$ ist immer positiv und nimmt den Wert 0 genau dann an, wenn die beiden Verteilungen $P$ und $R$ identisch sind. Diese Aussage formalisiert der nächste Satz.

**Satz 2.4.1** *Seien $P = \{p_1, \ldots, p_n\}$ und $R = \{r_1, \ldots, r_n\}$ zwei Wahrscheinlichkeitsverteilungen. Dann ist*

$$D(P||R) \geq 0,$$

*wobei die Gleichheit genau dann gilt, wenn $p_i = r_i \ \forall \ i \ \in \ \{1, \ldots, n\}$.*

**Beweis:** Siehe Hilfssatz 2.3.1 (1) mit

$$\begin{aligned} 0 &\leq -H(p_1, \ldots, p_n) - \sum_{i=1}^{n} p_i \log_2 r_i \\ 0 &\leq \sum_{i=1}^{n} p_i \log_2 p_i - \sum_{i=1}^{n} p_i \log_2 r_i \\ 0 &\leq \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{r_i} \end{aligned}$$

□

Da die Kullback Leibler Distanz nicht symmetrisch ist und die Dreiecksungleichung nicht erfüllt, handelt es sich hierbei nicht um eine echte Distanz zwischen zwei Wahrscheinlichkeitsverteilungen.

Im allgemeinen gilt also

$$D(P||R) = \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{r_i} \neq \sum_{i=1}^{n} r_i \log_2 \frac{r_i}{p_i} = D(R||P).$$

Nachdem wir nun die Kullback Leibler Distanz kennengelernt haben, wollen wir im nächsten Beispiel eine Anwendungsmöglichkeit anschauen.

**Beispiel:** Gegeben sei eine Quelle $Q$ mit Alphabet $A$ und einer zugehörigen Wahrscheinlichkeitsverteilung $P = \{p_1, p_2\}$ , $p_1 = 0.1$ , $p_2 = 0.9$.

Die Entropie von $P$ ist dann

$$\begin{aligned} H(p_1, p_2) &= -(0.1 \log_2 0.1 + 0.9 \log_2 0.9) \\ &= 0.469 \end{aligned}$$

Verwenden wir nun jedoch nicht die Verteilung $P$, sondern approximieren sie durch die Wahrscheinlichkeitsverteilung $R = \{r_1, r_2\}$ mit $r_1 = 0.25$ , $r_2 = 0.75$, so können wir die Distanz der eigentlichen Verteilung $P$ zur Verteilung $R$ mit der Kullback Leibler Distanz berechnen und erhalten

$$\begin{aligned} D(P||R) &= \frac{1}{10} \log_2 \left( \frac{1/10}{1/4} \right) + \frac{9}{10} \log_2 \left( \frac{9/10}{3/4} \right) \\ &= \frac{1}{10} \log_2 \left( \frac{4}{10} \right) + \frac{9}{10} \log_2 \left( \frac{12}{10} \right) \\ &= 0.105 \end{aligned}$$

Die Kullback Leibler Distanz hat in diesem Fall also den Wert 0.105 als Ergebnis. Addiert man zur Entropie von $P$ nun das Ergebnis der Kullback Leibler Distanz von $P$ zu $R$, so erhält man eine neue Abschätzung für die erwartete Kompression. Mit der Verteilung $P$ liegt die erwartete Codelänge bei 46.9 % der Länge der Nachricht, und durch Approximation der Verteilung $P$ durch $R$ also nur noch bei 57.4 %. Die erwartete Länge des Codes verschlechtert sich somit.

Dieses Ergebnis soll der folgende Satz formalisieren.

**Satz 2.4.2** *Sei eine Quelle $Q$ mit Alphabet $A$ und einer zugehörigen Wahrscheinlichkeitsverteilung $P = \{p_1, \ldots, p_n\}$ gegeben. Wird die Verteilung $P$ durch eine zweite Verteilung $R = \{r_1, \ldots, r_n\}$ approximiert, so gilt für die erwartete Länge $E(C)$ eines kompakten Codes $C$*

$$H(P) + D(P||R) \leq E(C) \leq H(P) + D(P||R) + 1.$$

**Beweis:** Analog zu Satz 2.3.1 gilt hier

$$-\log_2 r_i \leq l_i \leq -\log_2 r_i + 1.$$

Daraus folgt

$$\begin{aligned} \sum_{i=1}^{n} p_i l_i &\leq -\sum_{i=1}^{n} p_i \log_2 r_i + \sum_{i=1}^{n} p_i \\ &= -\sum_{i=1}^{n} p_i \log_2 r_i + \sum_{i=1}^{n} p_i + \sum_{i=1}^{n} p_i \log_2 p_i - \sum_{i=1}^{n} p_i \log_2 p_i \\ &= H(P) + D(P||R) + 1. \end{aligned}$$

□

An diesem Punkt stellt sich die Frage, wieso man eine gegebene Wahrscheinlichkeitsverteilung eines Modells durch eine andere Verteilung approximieren soll, wenn sich dabei die erwartete Kompression verschlechtert. Diese Frage ist auch berechtigt, jedoch kommen wir erst im späteren Verlauf dieser Arbeit auf sie zurück.

## 2.5 Codekompression

Bisher haben wir uns mit allgemeiner Datenkompression beschäftigt. Die zu codierenden Informationen sind jeweils als binäre Nachrichten, also als endlicher Bitstrom aus $A^*$ interpretiert worden. Dieses Modell funktioniert natürlich nicht nur für binäre Informationen, sondern auch für beliebige Alphabete. In diesem Fall ordnet man jedem Element des Alphabets eine eindeutige Binärdarstellung zu, und erhält auf diese Weise einen eindeutigen Bitstrom für jede mögliche Nachricht.

Wir wollen uns aber von jetzt an nur noch mit Codekompression beschäftigen. Das bedeutet, dass die zu komprimierenden Daten Programm-Code sind. Es handelt sich also von nun an bei einer Nachricht um ein ausführbares Computerprogramm. Darunter können wir uns kleine Assemblerprogramme oder auch in JAVA oder C++ programmierte Software vorstellen. Was das genau heisst, soll das folgende Beispiel verdeutlichen.

**Beispiel:** Gegeben sei ein 32-Bit Assembler mit einem zugehörigen Instruktionssatz. Betrachten wir die Befehlsfolge:

```
1: MOV R1, R0
2: ADD R0, R2
```

Die entsprechende Binärdarstellung der beiden Instruktionen sei wie folgt:

```
1: 11100001 10100000 00010000 00000000
2: 11001000 10001000 00000000 00100000
```

Die jeweilige 32-Bit Repräsentation der einzelnen Befehle können wir als Elemente aus $A^*$ auffassen. Wie schon in den vorherigen Abschnitten ist eine Nachricht definiert als Konkatenation von Elementen aus $A^*$. In diesem Fall erzeugen wir also eine Nachricht, indem wir die beiden Instruktionen aneinanderhängen.

In diesem Beispiel ist die zu codierende Nachricht $m$ also gegeben durch

$$m =$$
1110000110100000000100000000000011001000100010000000000000100000

Auf diese Art und Weise lässt sich ein Assemblerprogramm durch einen endlichen Bitstrom darstellen, den wir als Nachricht interpretieren.

Somit sind wir also in der Lage, jedes beliebige Computerprogramm als binäre Nachricht darzustellen. Mit einer zugehörigen Wahrscheinlichkeitsverteilung können wir schliesslich die Entropie bestimmen und damit die erwartete Codelänge, also die zu erwartende Kompression des Computerprogramms, abschätzen. Anschliessend kann unter Verwendung eines Codierverfahrens, wie beispielsweise der Huffman-Codierung oder der arithmetischen Codierung[1] und der entsprechenden Wahrscheinlichkeitsverteilung das Programm komprimiert werden.

Wir haben nun schon einige Grundlagen aus dem Bereich der Datenkompression kennengelernt. Dabei war eine Quelle $Q$ mit Alphabet $A$ und einer zugehörigen Wahrscheinlichkeitsverteilung $P$ stets gegeben. Eine Quelle dieser Art haben wir als erinnerungslose Quelle bezeichnet. Sie ist das einfachste Modell einer Quelle, die man zur Datenkompression verwenden kann.

Da sie aber im Folgenden nicht mehr ausreichend ist, müssen wir kompliziertere Modelle betrachten. Bevor wir das können, müssen wir zunächst die sogenannten *Markowprozesse* kennenlernen.

## 2.6 Markowprozesse

Zur Formalisierung der Markowprozesse benötigen wir zunächst einige Grundlagen aus der Stochastik. Diese Grundlagen, sowie ausführliche Informationen zu stochastischen Prozessen, können beispielsweise in den Büchern „Probability and Random Processes“ [GS01], von Geoffrey R. Grimmett und David R. Stirzaker oder „Einführung in die Wahrscheinlichkeitstheorie und Statistik“ [Kre00], von Ulrich Krengel, nachgelesen werden.

Zunächst ordnen wir einem Zufallsexperiment mit endlich vielen Ausgängen eine endliche nichtleere Menge $\Omega$ zu, deren Elemente $\omega$ die Versuchsausgänge bezeichnen. Sie heissen Ergebnisse oder auch Stichproben. $\Omega$ heisst Ergebnismenge oder gelegentlich auch Ergebnisraum. Die Teilmengen von $\Omega$ sind die Ereignisse. Wir identifizieren $X \subset \Omega$ mit dem Ereignis, dass ein $\omega \in X$ der Versuchsausgang ist. Das Komplement $X^C$ bezeichnet das Ereignis, dass $X$ nicht geschieht. Die Anzahl der Elemente in $X$ wollen wir mit $Card(X)$ bezeichnen.

Nun müssen den Ereignissen noch Wahrscheinlichkeiten zugeordnet werden. Damit wir eine Wahrscheinlichkeitsverteilung $P = \{p_1, \ldots, p_{|\Omega|}\}$ erhalten, muss natürlich gelten

$$\sum_{\omega \in \Omega} p(\omega) = 1.$$

**Definition 2.6.1** *Sei $\Omega$ eine endliche nichtleere Menge von Versuchsausgängen eines Zufallsexperiments und $P$ eine Wahrscheinlichkeitsverteilung, die den Elementen aus $\Omega$ ihre jeweilige Wahrscheinlichkeit zuordnet. Dann heisst das Paar $(\Omega, P)$ der dem Experiment zugeordnete Wahrscheinlichkeitsraum.*

Als nächstes benötigen wir den Begriff der *bedingten Wahrscheinlichkeit*. Sie ist folgendermassen definiert.

[1] Die Huffman- bzw. arithmetische Codierung kann im Buch „Introduction to Data Compression“ [Say00], von K. Sayood nachgelesen werden.

**Definition 2.6.2** *Sei $(\Omega, P)$ ein Wahrscheinlichkeitsraum. Für beliebige Ereignisse $Y \subset \Omega$ mit $P(Y) > 0$ ist die bedingte Wahrscheinlichkeit $P(X|Y)$ von $X \subset \Omega$ bei gegebenem $Y$ definiert durch*

$$P(X|Y) = \frac{Card(X \cap Y)}{Card(Y)}.$$

*Aus $P(X \cap Y) = Card(X \cap Y)/Card(\Omega)$ und $P(Y) = Card(Y)/Card(\Omega)$ ergibt sich die Gleichung*

$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)}.$$

Wenden wir uns nun den sogenannten *stochastischen Prozessen* zu. Das heisst, wir betrachten Familien von Zufallsvariablen $X_t, t \in \{0, 1, \ldots, m\}$, die die zeitliche Entwicklung eines Zufallsgeschehens beschreiben.[2]

**Definition 2.6.3** *Sei $(\Omega, P)$ ein Wahrscheinlichkeitsraum, $T = \{0, 1, \ldots, m\}$ eine endliche Indexmenge und $I$ eine endliche Ereignismenge. Eine Familie $\{X_t, t \in T\}$ von Zufallsvariablen mit Werten in $I$ heisst endlicher stochastischer Prozess mit Parameterbereich $T$ und Ereignisraum $I$.*

Mit Hilfe der Definition eines stochastischen Prozesses ist es jetzt möglich, die Markowsche Kette[3] zu formalisieren.

**Definition 2.6.4** *Eine endliche Markowsche Kette ist ein endlicher stochastischer Prozess $\{X_n, n \in T\}$ mit endlichem Ereignisraum $I$, der die folgende Markowsche Eigenschaft besitzt:*

*Für alle $n \in T$ und für alle $i_0, \ldots, i_{n+1} \in I$ mit*

$$P(X_0 = i_0, \ldots, X_n = i_n) > 0$$

*ist*

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n).$$

Wir interpretieren $X_n$ als Ereignis zum Zeitpunkt $n$. Der Prozess hat die Markowsche Eigenschaft, wenn die Wahrscheinlichkeit, dass zur Zeit $n + 1$ das Ereignis $X_{n+1}$ eintritt, nur vom Ereignis $X_n$ zum Zeitpunkt $n$ abhängt, aber nicht davon, welche Ereignisse zu früheren Zeitpunkten eingetreten sind.

Die erste Bedingung der Definition 2.6.4 soll nur garantieren, dass die bedingten Wahrscheinlichkeiten wohldefiniert sind. Im Folgenden werden wir auf sie verzichten.

[2] Im Rahmen meiner Diplomarbeit möchte ich mich auf endliche stochastische Prozesse sowie endliche Markowsche Ketten beschränken, da diese für den weiteren Verlauf meiner Arbeit vollkommen ausreichend sind. In der von mir angegebenen Literatur findet man natürlich die allgemein gültigen Definitionen, jedoch benötigt man hierfür wesentlich mehr stochastische Grundlagen, auf die ich an dieser Stelle verzichten möchte.

[3] Andrej Andrejewitsch Markow (1856-1922).

Bei einer Markowschen Kette beeinflusst also das Ereignis zum jetzigen Zeitpunkt das unmittelbar nächste Ereignis in der Zukunft. Man spricht in diesem Fall auch von einer Markowschen Kette erster Ordnung. Diese Bezeichnung wollen wir auch beibehalten. Wenn nun bei einer Markowschen Kette erster Ordnung das jetzige Ereignis ausschliesslich vom vorherigen Ereignis abhängt, stellt sich unmittelbar die Frage, wie sich das bei einer Markowschen Kette höherer Ordnung verhält. Diese Frage werden uns die nächsten Definitionen beantworten.

**Definition 2.6.5** *Eine endliche Markowsche Kette zweiter Ordnung ist ein endlicher stochastischer Prozess* $\{X_n, n \in T\}$ *mit endlichem Ereignisraum* $I$*, der die folgende Eigenschaft besitzt:*

*Für alle* $n \in T \setminus \{0\}$ *und für alle* $i_0, \ldots, i_{n+1} \in I$ *ist*

$$P(X_{n+1} = i_{n+1}|X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1}|X_n = i_n, X_{n-1} = i_{n-1}).$$

Aus dieser Definition sehen wir, dass bei einer Markowschen Kette zweiter Ordnung die Ereignisse $X_n$ und $X_{n-1}$ das Ereignis $X_{n+1}$ beeinflussen. Das jeweils nächste Ereignis hängt also nicht nur vom vorherigen, sondern auch noch vom Ereignis davor ab.

Nun ist es wohl offensichtlich, wie sich aus den Definitionen 2.6.4 und 2.6.5 eine allgemeine Formulierung für Markowsche Ketten beliebiger Ordnung ableiten lässt.

**Definition 2.6.6** *Eine endliche Markowsche Kette k-ter Ordnung ist ein endlicher stochastischer Prozess* $\{X_n, n \in T\}$ *mit endlichem Ereignisraum* $I$*, der die folgende Eigenschaft besitzt:*

*Für alle* $n \in T \setminus \{0, \ldots, k-2\}$ *und für alle* $i_0, \ldots, i_{n+1} \in I$ *ist*

$$P(X_{n+1} = i_{n+1}|X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1}|X_n = i_n, \ldots, X_{n-k+1} = i_{n-k+1}).$$

Das jeweilige nächste Ereignis eines Prozesses wird bei einer Markowschen Kette $k$-ter Ordnung somit von den $k$ Ereignissen zuvor beeinflusst. Die Ordnung gibt also an, von wievielen Ereignissen zuvor, das nächste Ereignis abhängt.

Beschäftigen wir uns als nächstes mit der Entropie von Markowprozessen. Hierzu benötigen wir zunächst den Begriff der *bedingten Entropie*.

**Definition 2.6.7** *Seien* $X$ *und* $Y$ *zwei Zufallsvariablen mit* $X = \{x_1, \ldots, x_s\}$ *sowie* $Y = \{y_1, \ldots, y_s\}$*. Die bedingte Entropie* $H(Y|X)$ *ist definiert als*

$$\begin{aligned} H(Y|X) &= \sum_{i=1}^{s} p(x_i) \cdot H(Y|X = x_i) \\ &= -\sum_{i=1}^{s} p(x_i) \cdot \sum_{j=1}^{s} p(y_j|x_i) \log_2 p(y_j|x_i) \\ &= -\sum_{i=1}^{s} \sum_{j=1}^{s} p(x_i, y_j) \log_2 p(y_j|x_i). \end{aligned}$$

Nun haben wir $m$ Zufallsvariablen eines endlichen stochastischen Prozesses gegeben. Die Entropie eines solchen Prozesses ist dann wie folgt definiert.

**Definition 2.6.8** *Die Entropie eines endlichen stochastischen Prozesses $\{X_i\}, i \in T$ ist definiert durch*

$$\mathcal{H} \;=\; \lim_{m\to\infty} H(X_m|X_{m-1}, X_{m-2}, \ldots, X_0).$$

Verwendet man nun Markow-Ketten als Modell zur Datenkompression, so entspricht die erwartete Codelänge eines Codes $C$ der Entropie der Markowschen Kette. Diese Aussage formalisieren die folgenden Definitionen.

**Definition 2.6.9** *Eine Wahrscheinlichkeitsverteilung $\mu = (\mu_1, \ldots, \mu_{|I|})$ heisst stationäre Verteilung der Markowschen Kette erster Ordnung, wenn gilt:*

$$\mu_s = \sum_{i=1}^{|I|} \mu_i \cdot P(X_{n+1} = s|X_n = i) \quad \forall\, s \in \{1, \ldots, |I|\}$$

**Definition 2.6.10** *Eine Markowsche Kette erster Ordnung heisst stationäre Markowsche Kette erster Ordnung, wenn für alle $i, j \in I$ $\;P(X_{n+1} = j|X_n = i)$ unabhängig von $n$ ist.*

**Definition 2.6.11** *Die Entropie einer stationären Markowschen Kette erster Ordnung $\{X_i\}, i \in T$ ist definiert durch*

$$\mathcal{H}_{M^1} \;=\; \lim_{m\to\infty} H(X_m|X_{m-1}, X_{m-2}, \ldots, X_0) \;=\; \lim_{m\to\infty} H(X_m|X_{m-1}) \;=\; H(X_1|X_0),$$

*wobei die bedingte Entropie $H(X_1|X_0)$ anhand der stationären Verteilung $\mu$ berechnet wird.*

Aus dieser Definition können wir die Definition für eine Markowsche Kette $k$-ter Ordnung wie folgt ableiten.

**Definition 2.6.12** *Die Entropie einer stationären Markowschen Kette k-ter Ordnung $\{X_i\}, i \in T$ ist definiert durch*

$$\begin{aligned}\mathcal{H}_{M^k} &= \lim_{m\to\infty} H(X_m|X_{m-1}, \ldots, X_0)\\ &= \lim_{m\to\infty} H(X_m|X_{m-1}, \ldots, X_{m-k})\\ &= H(X_k|X_{k-1}, \ldots, X_1, X_0),\end{aligned}$$

*wobei die bedingte Entropie $H(X_k|X_{k-1}, \ldots, X_1, X_0)$ anhand der stationären Verteilung $\mu$ berechnet wird.*

Einen Code $C$ für eine Markowsche Kette erster Ordnung können wir folgendermassen beschreiben.

**Definition 2.6.13** *Sei der Ereignisraum $I$ sowie eine stationäre Markowsche Kette erster Ordnung gegeben. Ein Code $C$ für eine stationäre Markowsche Kette erster Ordnung ist definiert durch*

$$C := \{C_s | s \in I\} \cup \{C_0\},$$

*wobei $C_0$ den Code für das erste Ereignis $X_0$ beschreibt.*

Es wird also jedem Element des Ereignisraums ein eigener Code zugewiesen. Je nachdem, welches Element $s \in I$ das Element $s' \in I$ zu einem Zeitpunkt $t \in \{1, \ldots, m\}$ bedingt, wird $s'$ eine entsprechende Codierung $c(s') \in C_{s'}$ zugewiesen.

Die erwartete Länge eines Codes $C$ lässt sich mit diesen Voraussetzungen dann wie folgt definieren.

**Definition 2.6.14** *Gegeben sei eine stationäre Markowsche Kette erster Ordnung mit stationärer Verteilung $\mu$. Für einen Code $C$ ist die erwartete Codelänge $E(C)$ dann definiert durch*

$$E(C) := \sum_{s \in I} \mu_s \sum_{x \in I} P(X_1 = x | X_0 = s) |c_{xs}|,$$

*wobei $|c_{xs}|$ die Länge des Codes für $X_1 = x$ beschreibt, wenn zuvor das Ereignis $s \in I$ eingetreten ist.*

Wie schon in Abschnitt 2.3 können wir auch hier einen Zusammenhang zwischen der Entropie des Markowschen Prozesses und der erwarteten Codelänge $E(C)$ eines Codes $C$ herleiten.

**Satz 2.6.1** *Gegeben sei eine stationäre Markowsche Kette erster Ordnung. Für die erwartete Codelänge $E(C)$ eines kompakten Codes $C$ gilt*

$$H(X_1|X_0) \leq E(C) \leq H(X_1|X_0) + 1.$$

**Beweis:** Analog zum Beweis des Satzes 2.3.1 gilt in diesem Fall

$$-\log_2 P(X_1 = x | X_0 = s) \;\leq\; |c_{xs}| \;\leq\; -\log_2 P(X_1 = x | X_0 = s) + 1,$$

wobei $|c_{xs}|$ wie in Definition 2.6.14 die Länge des Codes von $x \in I$ beschreibt, wenn zuvor das Ereignis $s \in I$ eingetreten ist. Mit $|c_{xs}| \leq -\log_2 P(X_1 = x | X_0 = s) + 1$ folgt dann:

$$\begin{aligned}
E(C) &= \sum_{s \in I} \mu_s \sum_{x \in I} P(X_1 = x | X_0 = s) \, |c_{xs}| \\
&\leq -\sum_{s \in I} \mu_s \sum_{x \in I} P(X_1 = x | X_0 = s) \log_2 P(X_1 = x | X_0 = s) \\
&\quad + \sum_{x \in I} P(X_1 = x | X_0 = s) \\
&= H(X_1|X_0) + 1
\end{aligned}$$

□

Die Definitionen 2.6.9, 2.6.10, 2.6.13 sowie 2.6.14 lassen sich auf Markowsche Ketten höherer Ordnung entsprechend erweitern, jedoch wollen wir an dieser Stelle darauf verzichten.

In den ersten Abschnitten dieses Kapitels haben wir uns eingehend mit Grundlagen aus dem Bereich der Datenkompression beschäftigt. Mit dem Modell der erinnerungslosen Quelle können wir unter Verwendung eines Codierverfahrens, beispielsweise der Huffman-Codierung oder auch der arithmetischen Codierung, beliebige Nachrichten aus $\{0, 1\}^*$ codieren.

Desweiteren haben wir gesehen, wie wir ausführbare Computerprogramme, also Programm-Code, als binäre Nachricht repräsentieren können.

Wir werden im Rahmen dieser Arbeit keine Programme komprimieren, sondern ihre Binärdarstellung für eine spätere Kompression analysieren.

Durch diese Analyse entstehen Wahrscheinlichkeitsverteilungen, von denen die Entropie bestimmt werden kann. Mit Hilfe der Entropie sind wir dann in der Lage, die erwartete Kompression abzuschätzen und somit die jeweilige Modellierung zu bewerten.

Da das Modell der erinnerungslosen Quelle hier aber nicht mehr ausreichend ist, wollen wir dieses Modell durch ein *Modell mit Erinnerungsvermögen* ersetzen. Aus diesem Grund haben wir uns in Abschnitt 2.6 mit Markowprozessen beschäftigt.

Im folgenden Kapitel werden wir nun sehen, wie Markowsche Ketten in der Daten-/Codekompression eine Anwendung finden.

# 3 Modelle

„Markowprozesse zur Modellierung bei Codekompression“ ist nicht nur der Titel dieser Diplomarbeit, sondern auch der Inhalt dieses Kapitels. Wir wollen uns jetzt mit einer anderen Art der Modellbildung zur Codekompression beschäftigen. Das grundlegende Modell einer Quelle, wie wir sie im zweiten Kapitel betrachtet haben, wird hierbei durch ein sogenanntes *Markow-Modell* ersetzt.

Als Grundlage dienen die Forschungsarbeiten „Random Access Decompression using Binary Arithmetic Coding“ [LW99a], „Code Compression for Embedded Systems“ [LW98] und „SAMC: A Code Compression Algorithm for Embedded Processors“ [LW99b] von Haris Lekatsas und Wayne Wolf.

Die Modellbildung, die in diesem Kapitel vorgestellt wird, unterscheidet sich allerdings etwas von der Modellbildung der genannten Arbeiten. Den Unterschied werden wir später beschreiben.

Weitere Informationen zu Markow-Modellen und ihrer Anwendung im Bereich der Datenkompression findet man in der Arbeit „Data Compression using Dynamic Markov Modelling“ [CH87], von Gordon V. Cormack und R. Nigel Horspool oder auch in dem Buch „Introduction to Data Compression“ [Say00], von Khalid Sayood.

Bei der folgenden Modellierung handelt es sich um ein *semi-adaptives Modell*, das eine gegebene Nachricht, in diesem Fall also die Binärdarstellung eines Computerprogramms, in einem 2-Phasen-Prozess komprimiert. Im ersten Schritt wird die Nachricht statistisch analysiert. Die charakteristischen Merkmale, also die Struktur, wird hier mit stochastischen Prozessen untersucht und auf diese Weise das Markow-Modell aufgebaut.

Im zweiten Schritt folgt dann die eigentliche Kompression, wobei das komprimierte Programm und das Markow-Modell für die spätere Decodierung gespeichert werden. Haris Lekatsas und Wayne Wolf haben in ihren Forschungsarbeiten immer die tabellenbasierte arithmetische Codierung von Howard und Vitter[4] als Kompressionsverfahren verwendet. Alternativ könnte man auch eine Huffman-Codierung verwenden, jedoch müsste man dabei die kompletten Huffman-Tabellen abspeichern, was mit wachsender Modellgrösse sehr komplex würde.

Die arithmetische Codierung ist hier effizienter, da sie, neben dem Verfahren selbst, im Wesentlichen nur die Wahrscheinlichkeitsverteilungen benötigt. Allerdings ist die arithmetische Codierung schwieriger zu implementieren, da sie exakte Arithmetik voraussetzt.

Die tabellenbasierte arithmetische Codierung von Howard und Vitter ersetzt die Arithmetik der arithmetischen Codierung durch sogenannte *table look-ups*. Hierbei wird die Codierung eines einzelnen Symbols aus einer Tabelle entnommen. Diese Version der arithmetischen Codierung lässt sich bei überschaubarer Tabellengrösse gut implementieren.

[4]Details zur tabellenbasierten arithmetischen Codierung sind in der Forschungsarbeit “Practical Implementations of Arithmetic Coding“ [PJ92], von P.G. Howard und J.S. Vitter nachzulesen.

Im weiteren Verlauf dieser Arbeit werden wir uns nun eingehend mit dem ersten Schritt der semi-adaptiven Modellierung, also der statistischen Analyse mittels Markowprozessen, beschäftigen. Zunächst haben wir also die Binärdarstellung eines Computerprogramms, wie in 2.5 beschrieben, zur Verfügung. Ein endlicher Bitstrom aus $\{0, 1\}^*$ repräsentiert ein zu komprimierendes Programm. Es existiert somit eine Nachricht $m$ der Form

$$m = \texttt{11100000110100010010100000110101011001000100010000110000...01101}.$$

In den nächsten Abschnitten wollen wir uns nun anschauen, auf welche verschiedene Art und Weise die Binärdarstellung von ausführbaren Programmen durch die Verwendung Markowscher Ketten modelliert werden kann.

## 3.1 Bit-Modelle

Beginnen wir mit dem einfachen Bit-Modell. Das Lesen eines einzelnen Bits der gegebenen Nachricht können wir in diesem Modell als Ereignis interpretieren. Eine Markowsche Kette erster Ordnung lässt sich auf die gegebene Nachricht wie folgt anwenden.

### 3.1.1 Markow-Kette erster Ordnung

Bei einer Markowschen Kette erster Ordnung im Bit-Modell bedingt der Wert des Bits zum aktuellen Zeitpunkt des Prozesses immer den Wert des Bits zum darauf folgenden Zeitpunkt. Der Wert des Bits zum Zeitpunkt $t$ bedingt also stets den Wert des Bits zum Zeitpunkt $t + 1, 0 \leq t \leq m - 1$. Beginnt nun der Prozess, so wird zunächst der Wert des ersten Bits der Nachricht unbedingt gelesen. Dieser Wert bedingt nun den Wert des nächsten Bits der Nachricht, der seinerseits wiederum den Wert des folgenden Bits bedingt.

Beim letzten Ereignis, also dem Lesen des letzten Bits der Nachricht, bedingt der Wert des vorletzten Bits.

Schauen wir uns das einmal in einer Grafik an.

$\boxed{0}\boxed{1}\,$001010110000010100101001010110101010...10

Das grau hinterlegte Bit ist das erste Bit der Nachricht und somit das Bit, das zum ersten Zeitpunkt des Prozesses unbedingt gelesen wird. Als zweites Ereignis der Markowschen Kette wird das zweite Bit, bedingt durch 0, gelesen. Im dritten Zeitpunkt der Markowschen Kette bedingt die 1 die folgende 0.

0 $\boxed{1}\boxed{0}$ 01010110000101001010010101101010...10

Der vierte Zeitpunkt des Prozesses - Hier bedingt die grau hinterlegte 0 das folgende Bit, in diesem Fall ebenfalls die 0.

01 $\boxed{0}\boxed{0}$ 1010110000101001010010101101010...10

Iteriert man diesen Prozess, so endet die Markowsche Kette mit dem letzten Ereignis, also 1 bedingt 0.

01001010110000101001010010101101010... $\boxed{1}\boxed{0}$

Abbildung 1: Bit-Modell - Markow-Kette erster Ordnung

Da jeweils der Wert eines Bits den Wert des unmittelbar nächsten Bits bedingt und ein Bit ein Element aus $\{0, 1\}$ ist, erhalten wir in diesem Modell zwei Wahrscheinlichkeitsverteilungen mit je zwei Elementen. Zum einen kann eine Null von Eins oder von Null gefolgt werden, zum anderen kann die Eins von Null oder von Eins gefolgt werden.

Aus dieser Abbildung können wir sehen, dass für alle Zeitpunkte $n \in T$ des obigen Prozesses und Zufallsvariablen $X_n$ mit Werten in $I = \{0, 1\}$ die Definition aus 2.6.4 gilt mit

$$P(X_{n+1} = i_{n+1}|X_0 = i_0, \dots, X_n = i_n) = P(X_{n+1} = i_{n+1}|X_n = i_n).$$

Es handelt sich hierbei also um eine Markowsche Kette erster Ordnung.

Zählt man während des Prozesses nun die entsprechenden Ereignisse, also beispielsweise Null folgt Null $n-$Mal, Eins folgt Null $m-$Mal, sowie Null folgt Eins $x-$Mal und Eins folgt Eins $y-$Mal, so erhält man die zugehörigen Wahrscheinlichkeitsverteilungen.

**Beispiel:** Sei eine Nachricht $m$ gegeben durch $m = 00011000$. Verwenden wir nun eine Markow-Kette erster Ordnung, so errechnen wir folgende Wahrscheinlichkeitsverteilungen:
Wahrscheinlichkeitsverteilung zum bedingenden Bit ’0’:

$$\begin{aligned} 0 \text{ bedingt } 0: \; 4/5 &= 80\% \\ 0 \text{ bedingt } 1: \; 1/5 &= 20\% \end{aligned}$$

Wahrscheinlichkeitsverteilung zum bedingenden Bit ’1’:

$$\begin{aligned} 1 \text{ bedingt } 0: \ 1/2 &= 50\% \\ 1 \text{ bedingt } 1: \ 1/2 &= 50\% \end{aligned}$$

Als nächstes wollen wir uns die graphische Repräsentation dieses Modells anschauen. Man spricht hierbei von dem sogenannten *Markow-Graph*.

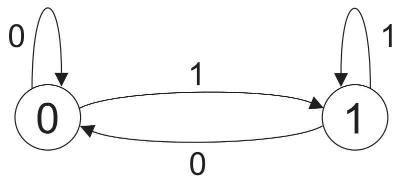



Abbildung 2: Bit-Modell - Markow-Graph erster Ordnung

Dieser Graph modelliert eine Markowsche Kette erster Ordnung im Bit-Modell. Merkt man sich an den Knoten 0 und 1 die Anzahl der jeweiligen Besuche, so erhält man die Häufigkeit, wie oft der Prozess in dem entsprechenden Zustand war. Die relativen Häufigkeiten der gewählten Ausgangskanten eines Knotens bilden Wahrscheinlichkeitsverteilungen. In diesem Fall erhalten wir für zwei Knoten mit je zwei Ausgangskanten genau zwei Wahrscheinlichkeitsverteilungen mit je zwei Elementen.

**Beispiel:** Der Markow-Graph sei nach dem Lesen der Nachricht 00011001110111111 wie folgt gegeben:

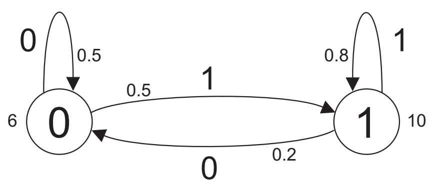



Abbildung 3: Beispiel eines Markow-Graphen erster Ordnung

In diesem Beispiel war der Prozess 6-mal im Zustand 0 und 10-mal im Zustand 1. Im Zustand 0 befindend wurde 3-mal die Kante 0 gewählt und 3-mal die Kante 1. Im Zustand 1 2-mal die Kante 0 und 8-mal die Kante 1. Daraus erhalten wir die Wahrscheinlichkeitsverteilungen mit $P_0 := \{0.5, 0.5\}$ sowie $P_1 := \{0.2, 0.8\}$.

Nach den Definitionen 2.6.7 und 2.6.11 können wir die Entropie dieses Prozesses ausrechnen. Es ergeben sich folgende Werte:

$$\begin{aligned} H_{P_0} &= 0.375 \\ H_{P_1} &= 0.4512 \end{aligned}$$

Für die Entropie der Markowschen Kette gilt dann:

$$\begin{aligned} \mathcal{H}_{M^1} &= H_{P_0} + H_{P_1} \\ &= 0.8262 \end{aligned}$$

In diesem Fall hat die Entropie den Wert 0.8262. Das bedeutet, dass man mit dieser Modellierung, also einer Markowschen Kette erster Ordnung im Bit-Modell, die gegebene Nachricht auf ≈ 83 % ihrer Länge codieren kann. Da die erwartete Codelänge der Entropie der Markow-Kette entspricht, ist mit dieser Modellierung keine bessere Kompression zu erreichen.

Somit ist eine Veränderung des Modells die einzige Möglichkeit, die Entropie zu verbessern.

An dieser Stelle wollen wir nun auf den Unterschied dieser Modellierung gegenüber der Modellierung von Lekatsas und Wolf eingehen.

In den Forschungsarbeiten [LW99a] und [LW99b] von Haris Lekatsas und Wayne Wolf hat der Modellgraph die folgende Form.

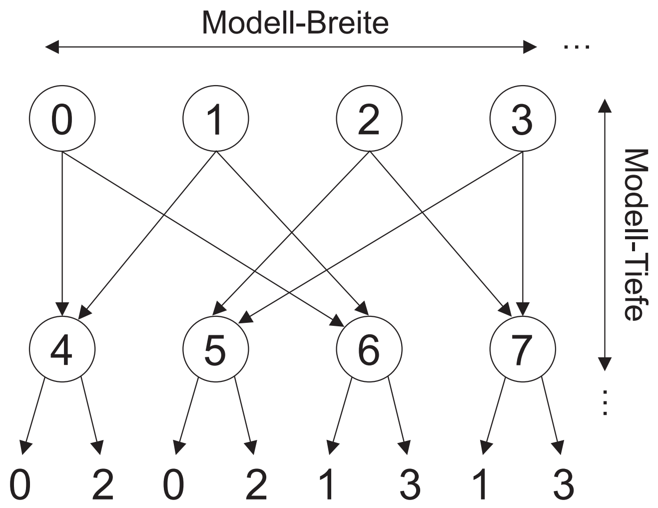



Abbildung 4: Positionsabhängige Modellierung

Jeder der Knoten besitzt zwei Ausgangskanten, wobei die linke Kante jeweils das Lesen einer Null und die rechte Kante das Lesen einer Eins repräsentiert. Das Modell wird im wesentlichen durch die Modell-Tiefe und die Modell-Breite beschrieben. Die Modell-Tiefe entspricht der Anzahl der Layer des Modells, wobei jeder Layer zu genau einem Bit der Nachricht korrespondiert. Die Modell-Breite beschreibt die Anzahl der Markow-Knoten pro Layer. Man beachte, dass der letzte Layer mit dem ersten Layer verbunden ist[5].

[5] Hier angedeutet durch die Knotennummer, auf die die jeweilige Kante zeigt.

Ist nun eine Nachricht aus $\{0, 1\}^*$ gegeben, so wird, beginnend beim Knoten 0, der Graph entsprechend durchlaufen. Hieraus resultieren dann die Wahrscheinlichkeitsverteilungen für die aus einem Knoten ausgehenden Kanten.

Für die Konkatenation 000 werden, beginnend beim Startknoten 0, die Knoten 4, 0 und 4 besucht. Um nun den Unterschied zu unserer Modellierung zu verdeutlichen, überlegen wir uns, dass wir bei der Konkatenation 000 immer im Knoten 0 bleiben[6].

Zum Zeitpunkt, wo die erste Null die zweite Null bedingt, berechnet dieses Modell eine andere Wahrscheinlichkeitsverteilung als zum Zeitpunkt, wo die zweite Null die dritte Null bedingt. In unserer Modellierung wird dieser Vorgang nicht unterschieden.

Diese Modellierung kann man als positionsabhängige Modellierung auffassen. Je nachdem, wie man die Modell-Tiefe und Modell-Breite wählt, erzeugt man eine Positionsabhängigkeit über $\log_2(Modell - Breite)$ Schritte. Die Konkatenation 000 belegt diese Eigenschaft, da für die dritte Null die gleiche Kante gewählt wird, wie für die erste Null.

Mit dieser Art der Modellierung ist natürlich eine bessere Entropie zu erwarten, jedoch besitzt dieses Modell eine viel höhere Anzahl an Wahrscheinlichkeitsverteilungen. Lekatsas und Wolf modellieren zudem ausschliesslich auf Bitbasis.

Bei unserer Modellierung verzichten wir auf die Positionsabhängigkeit, da für die noch folgenden Modelle die Komplexität viel zu gross werden würde.

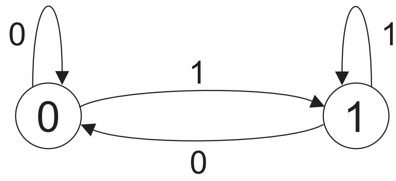
Bevor wir weitere Modellierungsmöglichkeiten im Bit-Modell betrachten, wollen wir uns kurz die zweite Phase des semi-adaptiven Modells anschauen.

### 3.1.2 Zweite Phase - Kompressionsvorgang

Ist mit der Modellierung aus 3.1.1 eine Nachricht aus $\{0, 1\}^*$ auf ihre Struktur analysiert, also die erste Phase des semi-adaptiven Modells abgeschlossen, so kann anhand des konstruierten Modells die binäre Nachricht codiert werden. Als Codierverfahren eignen sich beispielsweise die Huffman-Codierung und die arithmetische Codierung.

Für den konkreten Codiervorgang betrachten wir noch einmal den Markow-Graph aus dem vorherigen Abschnitt.

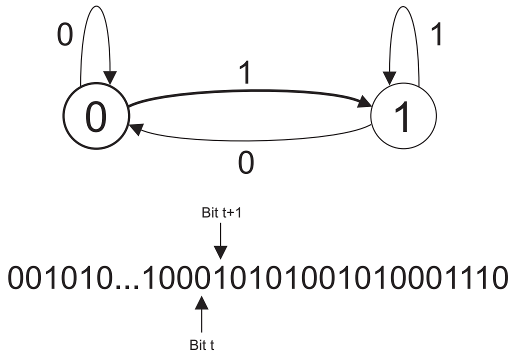
[6]Siehe Graph in Abbildung 2




Um nun die Codierung zu berechnen, wird die gegebene Nachricht ein zweites Mal gelesen. Nach jedem gelesenen Bit befindet sich der Prozess in einem der beiden Knoten 0 und 1. Diese repräsentieren im Kompressionsvorgang nun Code, der Schritt für Schritt erweitert wird.

Mit anderen Worten, sind $t$ Bits der gegebenen Nachricht gelesen, so existiert eine Codierung für diese $t$ Bits, die im nächsten Schritt um die Codierung des nächsten Bits erweitert wird. Anhand der in der ersten Phase berechneten Wahrscheinlichkeitsverteilungen wird entschieden, welcher Code der Codierung des nächsten Bits zugewiesen wird.

Skizzierung für den Zeitpunkt $t + 1$:




Hier sei $C_t$ die Codierung der Nachricht bis zum Bit $t$ und $C(t + 1)$ die Codierung, die Bit $t + 1$ zugewiesen wird. Es gilt

$$C_{t+1} = C_t \circ C(t + 1),$$

wobei das Zeichen $\circ$ die Konkatenation repräsentiert.

Dieser Abschnitt soll verdeutlichen, dass eine Daten-Modellierung durch Markowprozesse, gerade unter der Betrachtung des zugehörigen Markow-Graphen, eine sinnvolle Modellbildung für eine Daten/-Codekompression liefert.

Kommen wir aber nun wieder zurück zur Modellierung. Nachdem wir in Abschnitt 3.1.1 die Markowsche Kette erster Ordnung für das Bit-Modell kennengelernt haben, wollen wir jetzt die Markowsche Kette zweiter Ordnung betrachten.

### 3.1.3 Markow-Kette zweiter Ordnung

Bei einer Markow-Kette zweiter Ordnung gilt nun nach Definition 2.6.5:
Für alle $n \in T \setminus \{0\}$ und für alle $i_0, \ldots, i_{n+1} \in I = \{0, 1\}$ ist

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}).$$

An der formalen Beschreibung können wir sehen, dass in diesem Modell der Wert des Bits zum Zeitpunkt $t + 1$ des Prozesses immer von den Werten der Bits zu den Zeitpunkten $t$ und $t - 1$, $1 \leq t \leq m - 1$, bedingt wird. In diesem Modell betrachten wir also immer zwei bedingende Bits.

Wie in 3.1.1 endet der Prozess mit dem letzten gelesenen Bit der Nachricht. Sehen wir uns die Veränderungen zur Markowschen Kette erster Ordnung graphisch an.

0 1 0 01010110000101001010010101010101...010

Die ersten beiden, grau hinterlegten, Bits bedingen zum dritten Zeitpunkt des Prozesses das dritte Bit, die Null. Beim vierten Zeitpunkt verschieben sich die Kästchen um genau eine Position nach rechts. Bit vier wird also von den Bits Nummer zwei und drei bedingt.

0 1 0 0 1010110000101001010010101010101...010

Zum fünften Zeitpunkt der Markowschen Kette bedingt die Bitkombination '00' die folgende '1'.

01 0 0 1 010110000101001010010101010101...010

Mit dem Ereignis '01' bedingt '0' endet der Prozess beim letzten Bit der gegebenen Nachricht.

01001010110000101001010010101010101... 0 1 0

Abbildung 5: Bit-Modell - Markow-Kette zweiter Ordnung

In diesem Modell ergeben sich vier Wahrscheinlichkeitsverteilungen, da ein Ereignis von den zwei vorherigen Ereignissen bedingt wird. Ein Bit wird in diesem Modell also von der Bitkombination ’00’, ’01’, ’10’ oder ’11’ bedingt. Das Lesen eines Bits des gegebenen Bitstroms ist weiterhin ein Ereignis der Markowschen Kette. Daraus folgt, dass alle vier Verteilungen bis zu zwei Elemente besitzen[7].

Der Markow-Graph für dieses Modell sieht wie folgt aus:

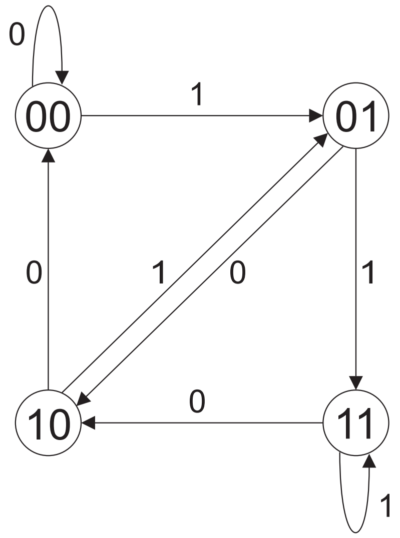



Abbildung 6: Bit-Modell - Markow-Graph zweiter Ordnung

Die Funktionsweise dieses Graphen ist analog zu dem in 3.1.1 dargestellten Graph. Aufgrund der Modelleigenschaft, dass zwei Bits das nächste Bit bedingen, haben wir vier verschiedene Knoten zu modellieren. Vergleicht man nun diesen Markow-Graphen mit dem aus 3.1.1, so stellt man fest, dass sich die Anzahl von Wahrscheinlichkeitsverteilungen, also die Anzahl an Knoten, verdoppelt hat. Schauen wir uns als nächstes an, ob sich diese Beobachtung auch für die Markowsche Kette dritter Ordnung feststellen lässt.

### 3.1.4 Markow-Kette dritter Ordnung

Nach Definition 2.6.6 können wir für eine Markowsche Kette dritter Ordnung die folgende Markowsche Eigenschaft definieren.

Für alle $n \in T \setminus \{0, 1\}$ und für alle $i_0, \dots, i_{n+1} \in I = \{0, 1\}$ ist

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \dots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}, X_{n-2} = i_{n-2}).$$

Bis auf die ersten drei Ereignisse wird in diesem Modell also ein Ereignis von den drei Ereignissen zuvor bedingt. In unserem Fall heisst das wiederum, dass wir

[7]Eine Verteilung muss nicht zwangsläufig zwei Elemente besitzen, da es in einer Nachricht vorkommen kann, dass beispielsweise die Bitkombination ’11’ ausschliesslich die ’0’ bedingt und niemals die ’1’. Dann hätte die zugehörige Verteilung $P_{11}$ nur ein Element.

jetzt drei bedingende Bits betrachten. Demzufolge werden die ersten drei Bits einer gegebenen Nachricht gelesen und bedingen das vierte Bit. Das fünfte Bit wird von den Bits zwei, drei und vier bedingt usw. Diesen Prozess wollen wir uns wieder in einer Grafik ansehen.

$\boxed{0\ 1\ 0}\boxed{0}$ 1010110000101001010010101010...1010

Im vierten Zeitpunkt der Markowschen Kette dritter Ordnung bedingen die Werte der ersten drei Bits der gegebenen Nachricht den Wert des vierten Bits. Die Bitkombination '010' bedingt also die '0'. Zum fünften Zeitpunkt bedingt die Kombination '100' die '1'.

0 $\boxed{1\ 0\ 0}\boxed{1}$ 010110000101001010010101010...1010

Im nächsten Schritt bedingen die Werte der Bits drei, vier und fünf den Wert des sechsten Bits. Also die Kombination '001' die '0'.

01 $\boxed{0\ 0\ 1}\boxed{0}$ 10110000101001010010101010...1010

Zum letzten Zeitpunkt der Markowschen Kette bedingen die Werte der Bits n-3, n-2 und n-1 den Wert des n'ten Bits.

01001010110000101001010010101010... $\boxed{1\ 0\ 1}\boxed{0}$

Abbildung 7: Bit-Modell - Markow-Kette dritter Ordnung

Betrachten wir nun alle möglichen Kombinationen, die man aus drei Bits konstruieren kann, so sehen wir, dass wir in diesem Modell acht Wahrscheinlichkeitsverteilungen erhalten. Hier bedingen also die Bitkombinationen '000', '001', '010',...,'111' ein Ereignis des Prozesses.

Der zugehörige Markow-Graph hat demnach folgende Gestalt.

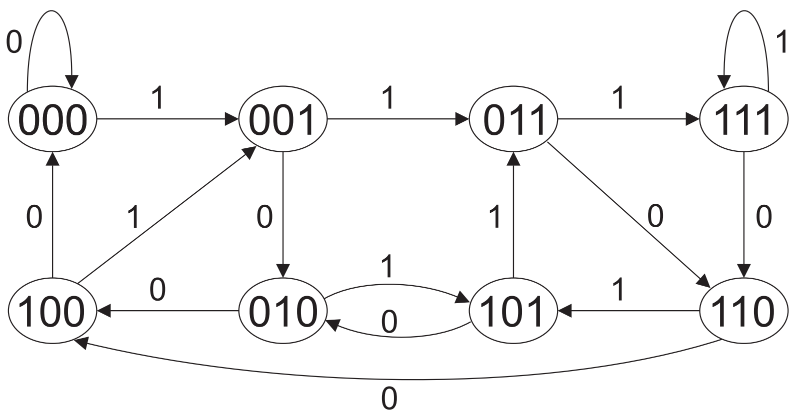



Abbildung 8: Bit-Modell - Markow-Graph dritter Ordnung

Aus dieser Abbildung sehen wir, dass die Anzahl der Knoten sich erneut verdoppelt hat. Durch Erhöhung der Ordnung des Markowschen Prozesses wächst also die Komplexität des Modells enorm an.

Aus den bisherigen drei Modellen sollte nun klar sein, wie sich der allgemeine Fall, also eine Markowsche Kette beliebiger Ordnung, ableiten lässt.

### 3.1.5 Markow-Kette höherer Ordnung

In diesem Abschnitt verallgemeinern wir unser Modell in eine Markowsche Kette der Ordnung $k$. Nach 2.6.6 gilt:

Für alle $n \in T \setminus \{0, \ldots, k-2\}$ und für alle $i_0, \ldots, i_{n+1} \in I = \{0, 1\}$ ist

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n, \ldots, X_{n-k+1} = i_{n-k+1}).$$

In einem Bit-Modell $k-$ter Ordnung wird der Wert eines Bits von den Werten der jeweiligen $k$ vorherigen Bits bedingt. Ausgenommen sind in diesem Fall natürlich die ersten $k$ Bits, die unbedingt gelesen werden. Die Grafik für den allgemeinen Fall sieht wie folgt aus:


Zunächst werden die ersten $k$ Bits einer Nachricht unbedingt gelesen. Diese $k$ Bits bedingen zum Zeitpunkt $k+1$ der Markowschen Kette $k$-ter Ordnung das Bit $k+1$. Für den Zeitpunkt $k+2$ des Prozesses verschieben sich die Kästchen um eine Position nach rechts. Nun bedingen also die Bits $2, \ldots, k+1$ das Bit $k+2$.

$$0\,\underbrace{\boxed{1\;0...1\;0}}_{\text{k-Bits}}\boxed{1}\,0000101001010001010100...10$$

Im Zeitpunkt $k + 3$ bedingen bei einer Markowschen Kette $k$-ter Ordnung die Bits $3, \ldots, k + 2$ das Bit $k + 3$.

$$01\,\underbrace{\boxed{0...1\;0\;1}}_{\text{k-Bits}}\boxed{0}\,000101001010001010100...10$$

Am Ende des Prozesses wird das $n$-te und letzte Bit der Nachricht von den Bits $n - k, \ldots, n - 1$ bedingt.

$$010...101000010100101000101010\,\underbrace{\boxed{1\;0\;0...1}}_{\text{k-Bits}}\boxed{0}$$

Abbildung 9: Bit-Modell - Markow-Kette beliebiger Ordnung

In den vorherigen Abschnitten haben wir gesehen, dass sich die Anzahl der Wahrscheinlichkeitsverteilungen verdoppelt, wenn man die Ordnung der Markowschen Kette um Eins erhöht.

Sei $\mathcal{P}$ nun die Menge aller Wahrscheinlichkeitsverteilungen einer Markowschen Kette $k$-ter Ordnung im Bit-Modell. Dann gilt:

$$|\mathcal{P}| = 2^k$$

Auf die Darstellung des Markow-Graphen wollen wir in diesem Fall verzichten.

Wir haben nun Markowsche Ketten unterschiedlicher Ordnung für das Bit-Modell kennengelernt. Dieses Modell, sowie die noch folgenden Modelle, habe ich im Rahmen meiner Diplomarbeit implementiert und an JAVA-Bytecode getestet.

Hierzu habe ich mehrere compilierte JAVA-Programmdateien der Software 'Star Office for Kids' in Dateien, die ausschliesslich aus Hexadezimalwerten bestehen, umgewandelt. Aus diesen Dateien wiederum kann man Binärdateien erstellen, die lediglich aus einem grossen Bitstrom bestehen. Dieser Bitstrom ist dann aber genau die Binärdarstellung des JAVA-Programms. Somit habe ich JAVA-Software als eine Nachricht aus $\{0, 1\}^*$ dargestellt.

Diese Dateiumwandlungen sowie alle anderen Programme meiner Diplomarbeit sind in Perl-Skripten realisiert[8].

Zum besseren Verständnis der Umwandlung der JAVA-Programme in eine binäre Nachricht ist der Vorgang hier noch einmal skizziert.

$$Dateiname.class \xrightarrow{f} Dateiname.hex \xrightarrow{g} Dateiname.bin$$

Die Funktionen $f$ und $g$ repräsentieren hier die entsprechenden Perl-Skripte, mit denen die Dateien erstellt sind.

Um die einzelnen Modelle vergleichen zu können, berechnen wir die Entropie der Wahrscheinlichkeitsverteilungen, die durch die Anwendung der Markowschen Kette einer bestimmten Ordnung resultieren.

Anhand der Entropie können wir dann abschätzen, welches Modell eine gute Kompression erwarten lässt, und welches Modell eine weniger gute Grundlage für eine Codekompression ist[9].

### 3.1.6 Testergebnisse

Die Testergebnisse für die verschiedenen Bit-Modelle sind in Tabelle 1 aufgelistet[10]. In der ersten Spalte steht der Name der Programmdatei, an der das Modell getestet wurde. Die zweite Spalte repräsentiert die jeweilige Ordnung der Markowschen Kette und die dritte Spalte zeigt den Wert der Entropie, also in etwa den Wert der zu erwartenden Kompression.

[8] Für Interessierte habe ich meine Programme zuzüglich verschiedener Testdateien auf eine CD gespeichert und dieser Arbeit beigefügt.

[9] Der Wert der Entropie sollte mindestens bei $\approx 0.75$ liegen. Eine Kompression auf mehr als 75 % der ursprünglichen Länge der Nachricht lohnt sich kaum noch und rechtfertig ihren Aufwand auch nicht mehr.

[10] Hier ist lediglich eine Auswahl der Testdateien aufgeführt. Die Tabelle beinhaltet die Dateien, die sowohl die beste Entropie, als auch die schlechteste Entropie meiner Tests lieferten. Eine grössere Auswahl an Testdateien sind auf der beigefügten CD zu finden.

| Testergebnisse der Bit-Modelle | | |
|---|---|---|
| Dateiname | Ordnung der Markow-Kette | Entropie |
| Application.bin | 1 | 0.9572 |
| | 2 | 0.9378 |
| | 3 | 0.9270 |
| | 4 | 0.9219 |
| | 5 | 0.9145 |
| | 6 | 0.9044 |
| | 7 | 0.8854 |
| | 8 | 0.8545 |
| Undo.bin | 1 | 0.9590 |
| | 2 | 0.9292 |
| | 3 | 0.8928 |
| | 4 | 0.8730 |
| | 5 | 0.8502 |
| | 6 | 0.8272 |
| | 7 | 0.7815 |
| | 8 | 0.7319 |
| TextParagraphSupplier.bin | 1 | 0.9119 |
| | 2 | 0.8847 |
| | 3 | 0.8577 |
| | 4 | 0.8519 |
| | 5 | 0.8416 |
| | 6 | 0.8252 |
| | 7 | 0.7926 |
| | 8 | 0.7665 |
| Cut.bin | 1 | 0.9582 |
| | 2 | 0.9287 |
| | 3 | 0.8922 |
| | 4 | 0.8728 |
| | 5 | 0.8502 |
| | 6 | 0.8262 |
| | 7 | 0.7809 |
| | 8 | 0.7299 |
| PictureChooser.bin | 1 | 0.9289 |
| | 2 | 0.9065 |
| | 3 | 0.8915 |
| | 4 | 0.8870 |
| | 5 | 0.8792 |
| | 6 | 0.8682 |
| | 7 | 0.8461 |
| | 8 | 0.8233 |

Tabelle 1: Testergebnisse der Bit-Modelle

In Tabelle 1 kann man sehr gut sehen, wie sich die jeweilige Entropie verbessert. Je höher die Ordnung der Markowschen Kette gewählt wird, desto besser wird die Entropie. Bei der Markowschen Kette erster und zweiter Ordnung liegt die Entropie durchschnittlich bei 0.93. Die entsprechenden Wahrscheinlichkeitsverteilungen kommen demnach der Gleichverteilung sehr nahe. Für eine Datenkompression ist das natürlich eine äusserst schlechte Voraussetzung.

Des weiteren können wir aus den Ergebnissen entnehmen, dass sich der Wert der Entropie, selbst bei einer Markowschen Kette achter Ordnung, kaum unter 0.75 befindet.

Die Testergebnisse zeigen also, dass die Bit-Modelle noch keine befriedigende Modellierung für eine Codekompression liefern. Nun stellt sich die Frage, wie wir die Modellierung verbessern können. Weitere Erhöhungen der Ordnung der Markowschen Kette werden nicht viel bringen. Zudem wächst die Anzahl der Wahrscheinlichkeitsverteilungen exponentiell an.

Allein aus Komplexitätsgründen können wir also die Ordnung der Markowschen Kette nicht beliebig erhöhen. Im Bit-Modell ergeben sich bei einer Markow-Kette achter Ordnung $2^8 = 256$ Wahrscheinlichkeitsverteilungen mit je zwei Elementen. Das wären 512 zu speichernde Fliesskommazahlen, die dann in der zweiten Phase von einem Codierverfahren verwendet werden, um die Nachricht zu codieren. Diese Zahl ist noch akzeptabel, jedoch darf sie nicht beliebig hoch werden. Gerade in eingebetteten Systemen, in denen Speicherplatz in der Regel sehr begrenzt ist, müssen Eigenschaften dieser Art besonders berücksichtigt werden.

Eine Verbesserung der Modellierung könnte man aber eventuell durch Erhöhung der Schrittweite erreichen. Bisher war ein Ereignis der Markowschen Kette immer das Lesen eines einzelnen Bits. Wenn wir nun diese Eigenschaft dahingehend verändern, dass pro Ereignis zwei Bits gelesen werden, so erhalten wir folgende Modelle.

## 3.2 2-Bit-Modelle

Im Gegensatz zu den in 3.1 vorgestellten Bit-Modellen definiert nun das Lesen von jeweils zwei Bits einer gegebenen Nachricht ein Ereignis der Markowschen Kette. Zur Vereinfachung der Beschreibung wollen wir zwei Bits auch als Bit-Paar bezeichnen. Eine Markowsche Kette erster Ordnung lässt sich in diesem Modell folgendermassen beschreiben.

### 3.2.1 Markow-Kette erster Ordnung

Der Wert des Bit-Paares zum Zeitpunkt $t + 1$ wird in dieser Modellierung jeweils vom Wert des Bit-Paares zum Zeitpunkt $t, 0 \leq t \leq m - 1$, bedingt. Wir betrachten in diesem Fall also Ereignisse der Form: '00' bedingt '01' etc.

Zum besseren Verständnis betrachten wir diesen Prozess in einer Grafik.

0 1 0 0 101011000010100101001010101010...1010

Zweiter Zeitpunkt der Markowschen Kette - Bit eins und zwei bedingen Bit drei und vier. Zum dritten Zeitpunkt wandern die Kästchen jetzt um zwei Positionen nach rechts. Bit drei und vier bedingen somit Bit fünf und sechs.

01 0 0 1 0 1011000010100101001010101010...1010

Zum vierten Zeitpunkt bedingen die Bits fünf und sechs die Bits sieben und acht. Die Kombination ’10’ bedingt also die Kombination ’10’.

0100 1 0 1 0 11000010100101001010101010...1010

Iteriert man diesen Vorgang bis zum Ende der Nachricht, so terminiert der Prozess. Zum letzten Zeitpunkt bedingen dann die Bits $n-3$ und $n-2$ die Bits $n-1$ und $n$.

0100101011000010100101001010101010... 1 0 1 0

Abbildung 10: 2-Bit-Modell - Markow-Kette erster Ordnung

Dieses Modell liefert uns vier Wahrscheinlichkeitsverteilungen mit je vier Elementen. Ein Ereignis ist nicht mehr das Lesen eines Bits, also das Lesen einer Null oder einer Eins, sondern das Lesen von ’00’, ’01’, ’10’ oder ’11’. In diesem Modell hat der Ereignisraum $I$ demnach vier Elemente.

Die prinzipielle Veränderung zum Bit-Modell ist also die Vergrösserung des zugrundeliegenden Ereignisraums. Bei der Markowschen Kette erster Ordnung im Bit-Modell bedingt ein Bit immer das nächste Bit. Hier bedingt nun ein Bit-Paar das folgende Bit-Paar. Daraus resultiert unmittelbar der entsprechende Markow-Graph dieses Modells.

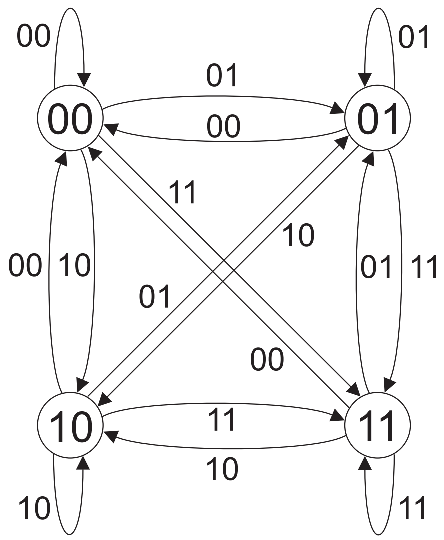



Abbildung 11: 2-Bit-Modell - Markow-Graph erster Ordnung

In dieser Abbildung können wir die Veränderung zum Bit-Modell noch einmal deutlich sehen. Im 2-Bit-Modell hat der Graph erster Ordnung bereits vier Knoten, die jeweils vier Eingangs- sowie vier Ausgangskanten besitzen.

Die Definition 2.6.4 lautet in diesem Fall:
Für alle $n \in T$ und für alle $i_0, \ldots, i_{n+1} \in I = \{00, 01, 10, 11\}$ ist

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n).$$

Auch in der formalen Beschreibung sehen wir, dass sich lediglich der Ereignisraum $I$ verändert hat.

**Beispiel:** Aus dem obigen Modell seien vier Wahrscheinlichkeitsverteilungen gegeben durch $P_{00} = \{0.1, 0.2, 0.3, 0.4\}$, $P_{01} = \{0.1, 0.1, 0.1, 0.7\}$, $P_{10} = \{0.25, 0.25, 0.2, 0.3\}$ und $P_{11} = \{0.1, 0.25, 0.35, 0.3\}$. Der Prozess ist 10-mal im Zustand 00, 17-mal im Zustand 01, 3-mal im Zustand 10 und 20-mal im Zustand 11 gewesen.

Nach 2.6.7 und 2.6.11 errechnet sich die Entropie des Markow-Prozesses wie folgt:

$$\begin{aligned} H_{P_{00}} &= 0.3692 \\ H_{P_{01}} &= 0.4610 \\ H_{P_{10}} &= 0.1191 \\ H_{P_{11}} &= 0.7532 \end{aligned}$$

Mit diesen Werten erhalten wir dann

$$\mathcal{H}_{M^1} = H_{P_{00}} + H_{P_{01}} + H_{P_{10}} + H_{P_{11}} = 1.7025$$

Das Ergebnis von 1.7025 bedeutet hier, dass zwei Bits der gegebenen Nachricht auf maximal 1.7025 Bits komprimiert werden können. Will man diesen Wert für ein Bit interpretieren, so dividiert man ihn durch Zwei. Um dieses Modell mit dem Bit-Modell vergleichen zu können, müssen wir die Entropie also halbieren. In diesem Fall ergibt sich mit diesem Modell eine erwartete Kompression auf 85.125 % der ursprünglichen Länge der Nachricht.

### 3.2.2 Markow-Kette zweiter Ordnung

Für eine Markowsche Kette zweiter Ordnung gilt nach Definition 2.6.5:

Für alle $n \in T \setminus \{0\}$ und für alle $i_0, \ldots, i_{n+1} \in I$ ist

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}).$$

Setzen wir wieder $I = \{00, 01, 10, 11\}$, so definieren wir die Markowsche Kette zweiter Ordnung für das 2-Bit-Modell.

Bei der Markowschen Kette zweiter Ordnung bedingen nun zwei Bit-Paare das jeweilige nächste Bit-Paar. Dementsprechend werden zu Beginn zwei Bit-Paare, also die ersten vier Bits der Nachricht, unbedingt gelesen. Diese beiden Paare bedingen dann das dritte Bit-Paar, das zusammen mit dem Zweiten wiederum das vierte Bit-Paar bedingt. Dieser Prozess stoppt mit dem Ereignis, das der Wert des letzten Bit-Paares vom Wert der zwei Bit-Paare zuvor bedingt wird.

Folgende Grafik skizziert den Prozessablauf einer Markowschen Kette zweiter Ordnung im 2-Bit-Modell:

0 1 0 0 | 1 0 | 10110000101001100101010...101010

Zeitpunkt drei des Prozesses: Bit-Paare eins und zwei bedingen Bit-Paar drei. Zum vierten Zeitpunkt fällt das erste Bit-Paar aus der Betrachtung heraus und Bit-Paar vier wird von Bit-Paar zwei und drei bedingt.

01 | 0 0 1 0 | 1 0 | 110000101001100101010...101010

Nachdem Bit-Paar vier von drei und zwei bedingt wurde, bedingt es zusammen mit Bit-Paar drei das fünfte Paar.

0100 $\boxed{1\ 0\ 1\ 0}\boxed{1\ 1}$ 000010100110010101010...101010

In dieser Grafik endet der Prozess mit dem Ereignis, das die Bitkombination ’1010’ die Kombination ’10’ bedingt.

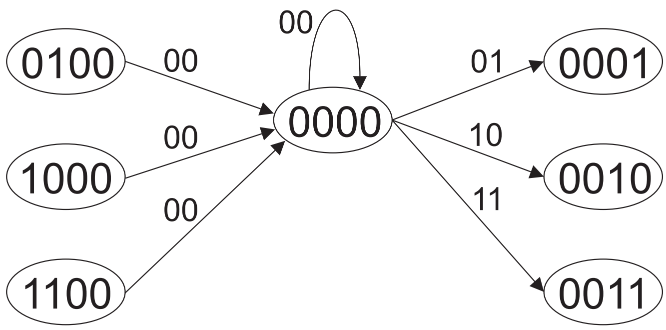
0100101011000010100110010101010... $\boxed{1\ 0\ 1\ 0}\boxed{1\ 0}$

Abbildung 12: 2-Bit-Modell - Markow-Kette zweiter Ordnung

Betrachtet man alle möglichen Kombinationen, die man durch vier Bits erzeugen kann, so erhält man $2^4 = 16$ verschiedene Elemente. Explizit sind das also die Bit-Kombinationen ’0000’, ’0001’, 0010’, . . ., ’1111’.

Das heisst, dass wir mit diesem Modell genau 16 Wahrscheinlichkeitsverteilungen erstellen. Wie im Modell erster Ordnung besitzen die Wahrscheinlichkeitsverteilungen bis zu vier Elemente.

Diese sowie die noch folgenden Markow-Graphen können wir aufgrund ihrer Komplexität nur noch ausschnittweise betrachten. In diesem Fall sind hier die Zustände ’0000’, ’0011’ und ’1010’ des Markow-Graph zweiter Ordnung im 2-Bit-Modell skizziert:



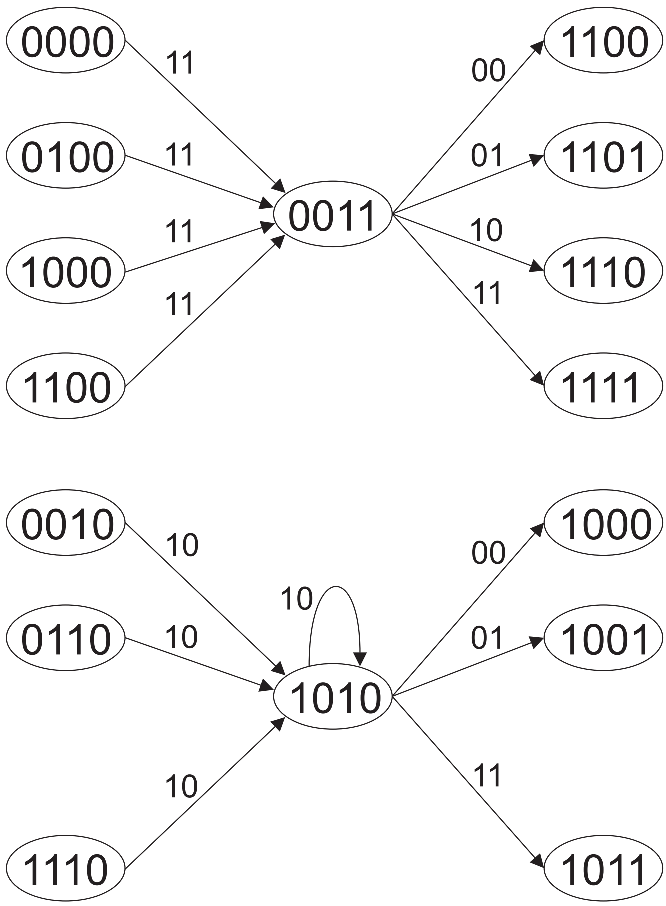



Abbildung 13: 2-Bit-Modell - Ausschnitte eines Markow-Graph zweiter Ordnung

Nachdem wir jetzt das 2-Bit-Modell mit Markowscher Kette erster und zweiter Ordnung kennengelernt haben, wollen wir das allgemeine 2-Bit-Modell mit Markow-Kette beliebiger Ordnung definieren.

### 3.2.3 Markow-Kette höherer Ordnung

Mit $I = \{00, 01, 10, 11\}$ gilt nach Definition 2.6.6 für eine Markowsche Kette $k$-ter Ordnung:

Für alle $n \in T \setminus \{0, \ldots, k-2\}$ und für alle $i_0, \ldots, i_{n+1} \in I = \{00, 01, 10, 11\}$ ist

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n, \ldots, X_{n-k+1} = i_{n-k+1}).$$

Bei der allgemeinen Formulierung wird das nächste Ereignis, also das nächste Bit-Paar, von den $k$ Bit-Paaren zuvor bedingt. Der Prozess im 2-Bit-Modell $k-$ter Ordnung lässt sich wie folgt skizzieren:

$$\underbrace{\boxed{0\ 1\ 0\ 1\ 1...1}}_{\text{2k-Bits}}\boxed{0\ 1}\ 001001010001010100...10$$

Zum Zeitpunkt $k + 1$ der Markowschen Kette bedingen die ersten $k$ Bit-Paare das Bit-Paar $k + 1$.

$$01\ \underbrace{\boxed{0\ 1\ 1...1\ 0\ 1}}_{\text{2k-Bits}}\boxed{0\ 0}\ 1001010001010100...10$$

Für jeden weiteren Zeitpunkt verschieben sich die Fenster um genau ein Bit-Paar.

$$0101\ \underbrace{\boxed{1...1\ 0\ 1\ 0\ 0}}_{\text{2k-Bits}}\boxed{1\ 0}\ 01010001010100...10$$

Der Prozess $k-$ter Ordnung terminiert mit dem Ereignis, das die Bit-Paare $n-k, \ldots, n-1$ das letzte Bit-Paar $n$ der Nachricht bedingen.

$$01011...101001001010001010\ \underbrace{\boxed{1\ 0\ 1\ 0\ 0...}}_{\text{2k-Bits}}\boxed{1\ 0}$$

Abbildung 14: 2-Bit-Modell - Markow-Kette der Ordnung k

Die Markowsche Kette erster Ordnung erzeugt in diesem Modell vier Wahrscheinlichkeitsverteilungen. Der Prozess zweiter Ordnung bereits derer sechzehn. Bei einer Markowschen Kette $k$-ter Ordnung betrachten wir, wie aus der obigen Abbildung hervorgeht, $2k$ bedingende Bits. Für die Anzahl der Wahrscheinlichkeitsverteilungen $|\mathcal{P}|$ gilt somit in diesem Modell:

$$|\mathcal{P}| = 2^{2k}$$

Erhöht man im 2-Bit-Modell also die Ordnung um Eins, so vervierfacht sich die Anzahl an unterschiedlichen Wahrscheinlichkeitsverteilungen des Modells. Im Bit-Modell resultieren aus einer Markowschen Kette achter Ordnung 256 Verteilungen. In diesem Modell erhalten wir bereits bei einem Prozess vierter Ordnung dieselbe Anzahl.

Nachdem wir nun das 2-Bit-Modell ausführlich betrachtet haben, wollen wir uns natürlich auch die Testergebnisse ansehen, die dieses Modell, unter Verwendung unterschiedlicher Ordnungen, hervorgebracht hat.

### 3.2.4 Testergebnisse

Die in der folgenden Tabelle aufgelisteten Testergebnisse sind mit denselben Dateien erstellt worden, wie die Testergebnisse des Bit-Modells.

Da die Anzahl der Wahrscheinlichkeitsverteilungen im 2-Bit-Modell, bei einer Markow-Kette vierter Ordnung, bereits 256 beträgt, und in diesem Fall schon 8 Bits ein Bit-Paar bedingen, sind hier nur Testergebnisse bis zur Markowschen Kette vierter Ordnung aufgeführt[11].

| **Testergebnisse der 2-Bit-Modelle** | | | |
|---|---|---|---|
| Dateiname | Ordnung der Markow-Kette | Entropie | Entropie pro Bit |
| Application.bin | 1 | 1.8419 | 0.9209 |
| | 2 | 1.8050 | 0.9025 |
| | 3 | 1.7393 | 0.8697 |
| | 4 | 1.5985 | 0.7993 |
| Undo.bin | 1 | 1.7742 | 0.8871 |
| | 2 | 1.6495 | 0.8248 |
| | 3 | 1.4865 | 0.7433 |
| | 4 | 1.2755 | 0.6378 |
| TextParagraphSupplier.bin | 1 | 1.7149 | 0.8574 |
| | 2 | 1.6549 | 0.8275 |
| | 3 | 1.5568 | 0.7784 |
| | 4 | 1.4173 | 0.7086 |
| Cut.bin | 1 | 1.7747 | 0.8874 |
| | 2 | 1.6512 | 0.8256 |
| | 3 | 1.4810 | 0.7405 |
| | 4 | 1.2660 | 0.6330 |
| PictureChooser.bin | 1 | 1.7770 | 0.8885 |
| | 2 | 1.7339 | 0.8669 |
| | 3 | 1.6629 | 0.8315 |
| | 4 | 1.5443 | 0.7722 |

Tabelle 2: Testergebnisse der 2-Bit-Modelle

Wie schon aus den Testergebnissen der Bit-Modelle hervorgeht, verbessert sich die Entropie von Ordnung zu Ordnung der Markowschen Kette. Diese Verbesserungen

[11] Man bedenke, dass im 2-Bit-Modell sechster Ordnung zwölf Bits ein Ereignis bedingen. Daraus resultieren dann aber auch $2^{12} = 4096$ verschiedene Wahrscheinlichkeitsverteilungen. Um also die Komplexität dieser Modelle halbwegs kontrollieren zu können, werden keine Markow-Ketten mit mehr als acht bedingenden Bits verwendet.

sind auch hier wieder zu sehen, allerdings deutlicher als beim Bit-Modell. Diese Beobachtung lässt sich durch die Veränderung des Ereignisraums, also der zu lesenden Bits pro Ereignis, begründen.

Die Dateien 'Undo.bin' und 'Cut.bin' weisen bei einer Markowschen Kette vierter Ordnung eine Entropie von $\approx 0.63$ vor. Dieses Ergebnis ist für eine spätere Kompression durchaus brauchbar. Im Gegensatz dazu, liegt die Entropie der Dateien 'Application.bin' sowie 'PictureChooser.bin' bei gleicher Ordnung über 0.75, was für eine effiziente Kompression zu viel ist.

Unter Verwendung der 2-Bit-Modelle haben sich die Ergebnisse aber trotzdem deutlich verbessert. Allerdings sind sie für eine Codekompression nur teilweise ausreichend. Demnach liefern die 2-Bit-Modelle nur gelegentlich eine gute Basis für eine Datenkomprimierung.

Die Verbesserung der Entropien der 2-Bit-Modelle gegenüber den Bit-Modellen begründet sich also durch die Erhöhung der Schrittweite pro Ereignis der Markowschen Kette. In den Bit-Modellen ist das Lesen eines einzelnen Bits ein Ereignis des Prozesses. In den 2-Bit-Modellen interpretieren wir ein Ereignis als das Lesen von zwei Bits der gegebenen Nachricht. Verändern wir nun diese Eigenschaft dahingehend, dass die Betrachtung von vier Bits ein Ereignis repräsentiert, so erhalten wir wieder eine neue Möglichkeit zur Charakterisierung gegebener Nachrichten.

## 3.3 Nibble-Modelle

Zunächst bezeichnen wir mit einem *Nibble* die Konkatenation von vier Bits, also einen 4-Bit-Block. In diesen Modellen entspricht das Lesen eines Nibble einem Ereignis. Eine Markowsche Kette erster Ordnung lässt sich im Nibble-Modell wie folgt anwenden.

### 3.3.1 Markow-Kette erster Ordnung

In dieser Modellierung wird der Wert des Nibble zum Zeitpunkt $t + 1$ vom Wert des Nibble zum Zeitpunkt $t, 0 \leq t \leq m - 1$ bedingt. Der Wert des ersten Nibble einer gegebenen Nachricht aus $\{0, 1\}^*$ wird also zum ersten Zeitpunkt des Prozesses unbedingt gelesen. Dieser Wert bedingt dann den Wert des folgenden Nibble, welcher wiederum den Wert des darauffolgenden Nibble der Nachricht bedingt.

Definieren wir die Menge $I$ mit $I = \{0000, 0001, 0010, \ldots, 1111\}$, so beschreibt die Definition 2.6.4 diesen Prozess. Es gilt also:

Für alle $n \in T$ und für alle $i_0, \ldots, i_{n+1} \in I = \{0000, 0001, 0010, \ldots, 1111\}$ ist

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n).$$

Eine Zufallsvariable $X$ nimmt also zum Zeitpunkt $n \in T$ einen der sechzehn verschiedenen Werte aus $I$ an. Betrachten wir diesen Prozess graphisch dargestellt.

Es sei wiederum die Binärdarstellung eines beliebigen Programm-Code gegeben.

0 1 0 0 1 0 1 0 110000101001...1010001100101010

Das grau hinterlegte, erste Nibble bedingt zum zweiten Zeitpunkt des Markowschen Prozesses das zweite Nibble. Zum dritten Zeitpunkt verschieben sich die Kästchen um genau ein Nibble, also vier Bit, nach rechts. Nibble zwei bedingt dann Nibble drei.

0100 1 0 1 0 1 1 0 0 00101001...1010001100101010

Im vierten Zeitpunkt bedingt das Nibble ’1100’ das Nibble ’0010’.

01001010 1 1 0 0 0 0 1 0 1001...1010001100101010

Wendet man dieses Verfahren nun bis zum Ende der Nachricht an, so terminiert der stochastische Prozess mit dem Ereignis, das die Bitkombination ’0010’ die Kombination ’1010’ bedingt.

0100101011000010100 1...10100011 0 0 1 0 1 0 1 0

Abbildung 15: Nibble-Modell - Markow-Kette erster Ordnung

In diesem Modell bedingen immer genau vier Bits ein Ereignis. Aus vier Bits lassen sich wiederum genau sechzehn verschiedene Bitkombinationen erzeugen. Daraus folgt, dass wir sechzehn unterschiedliche Wahrscheinlichkeitsverteilungen betrachten.

Die Elementanzahl pro Wahrscheinlichkeitsverteilung ergibt sich, wie schon in den Modellen zuvor, aus der Grösse des Ereignisraums. In diesem Fall besitzt eine Wahrscheinlichkeitsverteilung also bis zu sechzehn Elemente.

Die Veränderungen dieses Modells gegenüber dem 2-Bit-Modell finden sich auch im entsprechenden Markow-Graphen wieder. Aufgrund seiner Grösse sei hier nur ein Ausschnitt für den Zustand 1111 skizziert:

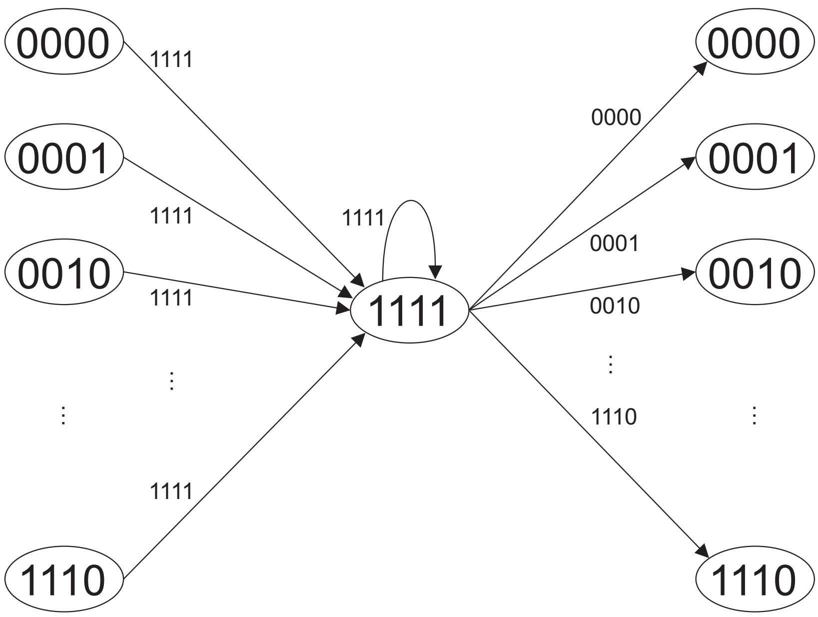



Abbildung 16: Nibble-Modell - Ausschnitt eines Markow-Graph erster Ordnung

Erweitern wir dieses Modell, indem wir die Ordnung der Markowschen Kette um Eins erhöhen.

### 3.3.2 Markow-Kette zweiter Ordnung

Für eine Markowsche Kette zweiter Ordnung gilt nach Definition 2.6.5:

Für alle $n \in T \setminus \{0\}$ und für alle $i_0, \ldots, i_{n+1} \in I$ ist

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}).$$

Mit $I := \{0000, 0001, 0010, \ldots, 1111\}$ erhalten wir die formale Beschreibung der Markowschen Kette zweiter Ordnung für das Nibble-Modell.

Das Ereignis zum Zeitpunkt $n$ wird also von den Ereignissen zum Zeitpunkt $n-1$ und $n-2$ beeinflusst[12]. Für dieses Modell bedeutet das, dass ein Nibble immer von den zwei vorherigen Nibble bedingt wird[13]. Schauen wir uns den Prozess ebenfalls in einer Grafik an.

[12] Ausgenommen sind hierbei natürlich die ersten beiden Ereignisse der Markowschen Kette.

[13] Anders ausgedrückt wird ein Nibble in diesem Modell immer vom vorherigen Byte, also den acht vorherigen Bits, bedingt.

In den ersten beiden Zeitpunkten des Markowschen Prozesses werden die ersten zwei Nibble der gegebenen Nachricht unbedingt gelesen.

$$\boxed{0\,1\,0\,0\,1\,0\,1\,0}\boxed{1\,1\,0\,0}\,101010011...0001100101010$$

Zum dritten Zeitpunkt bedingen die Werte der ersten zwei Nibble den Wert des dritten Nibble. Im vierten Zeitpunkt bedingt die Konkatenation ’10101100’ das Nibble ’1010’.

$$0100\,\boxed{1\,0\,1\,0\,1\,1\,0\,0}\boxed{1\,0\,1\,0}\,10011...0001100101010$$

Im fünften Ereignis wird ’1001’ von ’11001010’ bedingt.

$$01001010\,\boxed{1\,1\,0\,0\,1\,0\,1\,0}\boxed{1\,0\,0\,1}\,1...0001100101010$$

Die Markow-Kette endet mit dem Ereignis, dass das Nibble $n$ vom Nibble $n-1$ und $n-2$ bedingt wird.

$$010010101100101010011...0\,\boxed{0\,0\,1\,1\,0\,0\,1\,0}\boxed{1\,0\,1\,0}$$

Abbildung 17: Nibble-Modell - Markow-Kette zweiter Ordnung

In diesem Modell erhalten wir $2^8 = 256$ verschiedene Wahrscheinlichkeitsverteilungen. Da das Lesen eines Nibble weiterhin ein Ereignis dieses Prozesses ist, sich also der Ereignisraum nicht verändert hat, bestehen die Verteilungen aus jeweils sechzehn Elementen.

Das Nibble-Modell mit Markowscher Kette erster Ordnung liefert uns sechzehn verschiedene Verteilungen. Das gleiche Modell mit einer Markowschen Kette zweiter Ordnung bereits derer zweihundertsechsundfünfzig.

Der Markow-Graph dieses Modells besitzt also 256 Knoten mit jeweils 16 ausgehenden sowie 16 eingehenden Kanten. Für den Zustand 10001100 können wir den Graph folgendermassen skizzieren:

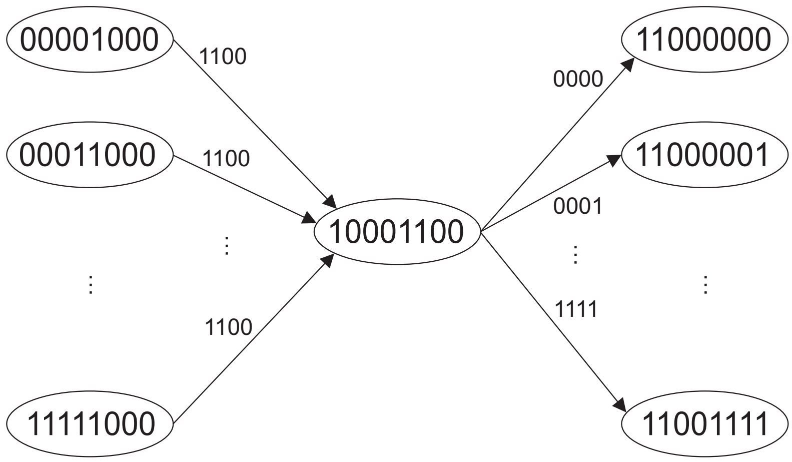



Abbildung 18: Nibble-Modell - Ausschnitt eines Markow-Graph zweiter Ordnung

An dieser Stelle sollte klar sein, wie das Nibble-Modell mit einer Markowschen Kette $k-$ter Ordnung funktioniert[14]. Für die Anzahl der Wahrscheinlichkeitsverteilungen erhalten wir

$$|\mathcal{P}| = 2^{4k}.$$

Kommen wir nun zu den Testergebnissen, die die Nibble-Modelle mit Markowscher Kette erster, sowie zweiter Ordnung ergeben haben.

### 3.3.3 Testergebnisse

Die folgende Tabelle beinhaltet die Testergebnisse der Dateien, die schon in den vorherigen Tabellen aufgeführt sind. Wie schon erwähnt, betrachten wir im Nibble-Modell nur Markowsche Ketten bis zur Ordnung Zwei.

[14] Da ich im Rahmen meiner Diplomarbeit das Nibble-Modell ausschliesslich bis zur Ordnung 2 verwendet habe, möchte ich auf die allgemeine Beschreibung einer Markowschen Kette k-ter Ordnung verzichten. In der nächsten Ordnung besitzt das Modell bereits $2^{12} = 4096$ Wahrscheinlichkeitsverteilungen mit je 16 Elementen. Das ergibt in der Summe 65536 zu speichernde Fliesskommazahlen. Diese Komplexität ist für eine spätere Codekompression völlig inakzeptabel.

| **Testergebnisse der Nibble-Modelle** | | | |
|---|---|---|---|
| Dateiname | Ordnung der Markow-Kette | Entropie | Entropie pro Bit |
| Application.bin | 1 | 3.3317 | 0.8329 |
| | 2 | 2.7918 | 0.6979 |
| Undo.bin | 1 | 2.8365 | 0.7086 |
| | 2 | 2.1164 | 0.5291 |
| TextParagraphSupplier.bin | 1 | 2.9786 | 0.7447 |
| | 2 | 2.4194 | 0.6048 |
| Cut.bin | 1 | 2.8344 | 0.7086 |
| | 2 | 2.1185 | 0.5296 |
| PictureChooser.bin | 1 | 3.1863 | 0.7966 |
| | 2 | 2.7158 | 0.6789 |

Tabelle 3: Testergebnisse der Nibble-Modelle

An den Ergebnissen können wir sehen, dass bereits die Markowsche Kette erster Ordnung Entropien deutlich unter 0.9 erzeugt. Die besten Ergebnisse liefert jedoch die Markowsche Kette zweiter Ordnung. In diesem Modell liegt die Entropie bei keiner getesteten Nachricht deutlich über 0.7. Mit anderen Worten, die Programmdateien lassen sich auf $\approx 70\ \%$ ihrer ursprünglichen Grösse komprimieren. Dies ist aber auch gleichzeitig ein eher schlechteres Einzelergebnis der Tests, da wir der Tabelle entnehmen können, dass sogar Entropien von ungefähr 0.55 aufgetreten sind.

Das Nibble-Modell mit einer Markowschen Kette zweiter Ordnung bietet demzufolge eine gute Modellierung der zu komprimierenden Nachrichten. Leider beinhaltet dieses Modell aber auch einen gravierenden Nachteil. Seine Anzahl an Wahrscheinlichkeitsverteilungen ist mit zweihundertsechsundfünfzig viel zu hoch. Zudem besitzt jede der Verteilungen bis zu sechzehn Elemente.

Im praktischen Fall muss demnach ein Modell mit 4096 Fliesskommazahlen gespeichert werden, um die Codekompression durchzuführen. Diese Komplexität ist gerade in eingebetteten Systemen deutlich zu gross.

Im nächsten Kapitel wollen wir nun verschiedene Methoden betrachten, die die Komplexität dieses Modells reduzieren, gleichzeitig aber so wenig Entropieverlust wie möglich verursachen.

# 4 Reduktion der Modellgrösse

In diesem Kapitel wollen wir ausschliesslich das Nibble-Modell mit Markowscher Kette zweiter Ordnung betrachten. Wie aus den Testergebnissen im dritten Kapitel hervorgeht, liefert uns diese Modellierung die besten Resultate. Ihr grosser Nachteil ist allerdings die hohe Anzahl an unterschiedlichen Wahrscheinlichkeitsverteilungen. Mit sechzehn Elementen pro Verteilung resultieren in diesem Modell 4096 zu speichernde Fliesskommazahlen.

Gerade in eingebetteten Systemen, wo nur wenig Speicherplatz zur Verfügung steht[15], lässt sich dieses Modell nicht anwenden. Genau in diesen Systemen findet es zusammen mit beispielsweise der tabellenbasierten arithmetischen Codierung von Howard und Vitter aber wiederum seine Anwendung.

Ziel muss es also sein, die Anzahl der zu speichernden Fliesskommazahlen zu reduzieren. Mit anderen Worten, die Anzahl der Wahrscheinlichkeitsverteilungen des Nibble-Modells mit Markowscher Kette zweiter Ordnung ist mit 256 viel zu gross und muss deutlich gesenkt werden.

Zu diesem Zweck wollen wir uns in den folgenden Abschnitten anschauen, auf welche Weise Wahrscheinlichkeitsverteilungen zusammengefasst werden können.

## 4.1 Konsolidierung von Wahrscheinlichkeitsverteilungen

Stellen wir uns vor, wir haben zwei Wahrscheinlichkeitsverteilungen $P = \{p_1, \dots, p_n\}$ und $Q = \{q_1, \dots, q_n\}$ des Nibble-Modells mit Markowscher Kette zweiter Ordnung, die wir zu einer Verteilung $U = \{u_1, \dots, u_n\}$ konsolidieren wollen.

Zu diesen beiden Verteilungen existiert die Häufigkeit, wie oft der Markowsche Prozess in dem zugehörigen Zustand war. Mit anderen Worten, es sei $s$ die Anzahl, wie häufig sich der Prozess im für die Verteilung $P$ bedingenden Zustand befand. Analog sei $s'$ die Häufigkeit, wie oft der Markowprozess im bedingenden Zustand für die Verteilung $Q$ war.

Damit wollen wir die relative Häufigkeit $\lambda$ definieren durch

$$\lambda := \frac{s}{s + s'}$$

und $1 - \lambda$ durch

$$1 - \lambda := \frac{s'}{s + s'}.$$

Mit diesen Vereinbarungen können wir nun ein Optimalitätskriterium für die Konsolidierung von Wahrscheinlichkeitsverteilungen definieren.

[15]Beispielsweise die Codekompression auf Smartcards.

**Definition 4.1.1** *Eine Wahrscheinlichkeitsverteilung $U = \{u_1, \ldots, u_n\}$ heisst optimal aus einer Verteilung $P = \{p_1, \ldots, p_n\}$ und einer Verteilung $Q = \{q_1, \ldots, q_n\}$ konsolidiert, wenn die Summe der gewichteten Kullback Leibler Distanzen von P und Q zu U minimal ist. Wenn also gilt:*

$$\lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{u_i} \; + \; (1-\lambda) \sum_{i=1}^{n} q_i \log_2 \frac{q_i}{u_i}$$

*ist minimal. $\lambda$ und $1-\lambda$ seien wie zuvor definiert.*

Die Motivation dieser Definition findet sich in der Eigenschaft, dass die Kullback Leibler Distanz von einer Verteilung $P$ zu einer Verteilung $U$ genau den Entropieverlust beschreibt, wenn man $P$ durch $U$ approximiert. Anders ausgedrückt, die Kullback Leibler Distanz repräsentiert hier die zusätzlich zu erwartende Codelänge, die wir durch Approximation von Wahrscheinlichkeitsverteilungen erhalten[16].

Der folgende Satz beschreibt nun, wie die Approximationsverteilung $U$ berechnet werden muss, um die zusätzlich zu erwartende Codelänge zu minimieren.

**Satz 4.1.1** *Es seien die Wahrscheinlichkeitsverteilungen $P = \{p_1, \ldots, p_n\}$ und $Q = \{q_1, \ldots, q_n\}$ mit den relativen Häufigkeiten $\lambda$ und $1-\lambda$ gegeben. Dann ist die Wahrscheinlichkeitsverteilung $U = \{u_1, \ldots, u_n\}$ optimal aus P und Q konsolidiert, wenn gilt*

$$u_i = \lambda \cdot p_i + (1-\lambda) \cdot q_i$$

*für alle $i \in \{1, \ldots, n\}$.*

**Beweis:** Für den Beweis dieses Satzes betrachten wir die Formel aus Definition 4.1.1

$$\lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{u_i} \; + \; (1-\lambda) \sum_{i=1}^{n} q_i \log_2 \frac{q_i}{u_i}.$$

Um nun die Aussage zu beweisen, muss die Funktion ihren minimalen Wert annehmen. Das heisst, die gewichtete Kullback Leibler Distanz von $P$ zu $U$ und von $Q$ zu $U$ muss als Summe möglichst gering sein. Es gilt:

$$\begin{aligned}
&\lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{u_i} \; + \; (1-\lambda) \sum_{i=1}^{n} q_i \log_2 \frac{q_i}{u_i} \; = \\
&\quad = \; \lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{u_i} \; + \; (1-\lambda) \sum_{i=1}^{n} q_i \log_2 \frac{q_i}{u_i} \; + \; \lambda \sum_{i=1}^{n} p_i \log_2 \frac{q_i}{u_i} \; - \; \lambda \sum_{i=1}^{n} p_i \log_2 \frac{q_i}{u_i} \\
&\quad = \; \lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \; + \; (1-\lambda) \sum_{i=1}^{n} q_i \log_2 \frac{q_i}{u_i} \; + \; \lambda \sum_{i=1}^{n} p_i \log_2 \frac{q_i}{u_i} \\
&\quad = \; \lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \; + \; \sum_{i=1}^{n} (\lambda\, p_i + (1-\lambda)\, q_i) \log_2 \frac{q_i}{u_i}
\end{aligned}$$

[16]Siehe auch Abschnitt 2.4

Mit $z_i = \mu \cdot p_i + (1-\mu) \cdot q_i \ \ \forall\, i \in \{1, \ldots, n\}$ gilt dann

$$\begin{aligned}
\lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \ &+ \ \sum_{i=1}^{n} (\lambda\, p_i + (1-\lambda)\, q_i) \log_2 \frac{q_i}{u_i} \\
&= \lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \ + \ \sum_{i=1}^{n} z_i \log_2 \frac{q_i}{u_i} \\
&= \lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \ + \ \sum_{i=1}^{n} z_i \log_2 \frac{1}{u_i} \ - \ \sum_{i=1}^{n} z_i \log_2 \frac{1}{q_i} \\
&= \lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \ + \ \sum_{i=1}^{n} z_i \log_2 \frac{z_i}{u_i} \ - \ \sum_{i=1}^{n} z_i \log_2 \frac{z_i}{q_i}.
\end{aligned}$$

Der Ausdruck

$$\lambda \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \ + \ \sum_{i=1}^{n} z_i \log_2 \frac{z_i}{u_i} \ - \ \sum_{i=1}^{n} z_i \log_2 \frac{z_i}{q_i}$$

ist minimal, wenn

$$\sum_{i=1}^{n} z_i \log_2 \frac{z_i}{u_i} = 0.$$

Dies ist nach Satz 2.4.1 genau dann der Fall, wenn $z_i = u_i \ \forall\, i \in \{1, \ldots, n\}$. Es folgt hier also $\mu = \lambda$ explizit.

□

Zwei Wahrscheinlichkeitsverteilungen werden somit optimal zu einer Wahrscheinlichkeitsverteilung zusammengefasst, indem die einzelnen Elemente komponentenweise und entsprechend gewichtet, addiert werden.

Aus diesem Beweis können wir nun den Beweis für die allgemeine Aussage ableiten.

Hierzu seien $P^1$,. . ., $P^m$ $m$ Wahrscheinlichkeitsverteilungen[17] mit $P^j = \{p^j_1, \ldots, p^j_n\}$. Desweiteren seien $\lambda^1, \ldots, \lambda^m$ die relativen Häufigkeiten der Verteilungen $P^1$,. . ., $P^m$ mit

$$\sum_{j=1}^{m} \lambda^j = 1.$$

Mit diesen Vereinbarungen können wir den folgenden Satz formalisieren.

[17] Lies: „P oben 1“ etc. - Diese Notation dient lediglich zur Unterscheidung der Elemente, es ist keine Potenzierung.

**Satz 4.1.2** *Es seien m Wahrscheinlichkeitsverteilungen $P^1, \dots, P^m$ gegeben mit $P^j = \{p_1^j, \dots, p_n^j\}$ $\forall\ j \in \{1, \dots, m\}$. Desweiteren seien $\lambda^1, \dots, \lambda^m$ die relativen Häufigkeiten der Verteilungen $P^1, \dots, P^m$ mit $\sum_{j=1}^m \lambda^j = 1$. Dann ist die Wahrscheinlichkeitsverteilung $U = \{u_1, \dots, u_n\}$ optimal aus $P^1, \dots, P^m$ konsolidiert, wenn gilt*

$$u_i = \sum_{j=1}^{m} \lambda^j \cdot p_i^j$$

*für alle $i \in \{1, \dots, n\}$.*

**Beweis:** Wir betrachten die Funktion

$$\sum_{j=1}^{m} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{u_i}.$$

Wie schon im Beweis des Satzes 4.1.1 muss auch hier die Funktion ihr Minimum annehmen. Wir erhalten:

$$\begin{aligned}
&\sum_{j=1}^{m} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{u_i} \quad = \\
&\quad = \sum_{j=1}^{m} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{u_i} + \sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{u_i} - \sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{u_i} \\
&\quad = \sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{p_i^m} + \sum_{j=1}^{m} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^m}{u_i} \\
&\quad = \sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{p_i^m} + \sum_{i=1}^{n} \left(\lambda^1\, p_i^1 + \lambda^2\, p_i^2 + \ldots + \lambda^m\, p_i^m\right) \log_2 \frac{p_i^m}{u_i}
\end{aligned}$$

Setzen wir

$$z_i = \sum_{j=1}^{m} \mu^j \cdot p_i^j, \quad 1 \le i \le n,$$

so bekommen wir daraus

$$\begin{aligned}
&\sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{p_i^m} + \sum_{i=1}^{n} \left(\lambda^1\, p_i^1 + \lambda^2\, p_i^2 + \ldots + \lambda^m\, p_i^m\right) \log_2 \frac{p_i^m}{u_i} \quad = \\
&\quad = \sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{p_i^m} + \sum_{i=1}^{n} z_i \log_2 \frac{p_i^m}{u_i} \\
&\quad = \sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{p_i^m} + \sum_{i=1}^{n} z_i \log_2 \frac{1}{u_i} - \sum_{i=1}^{n} z_i \log_2 \frac{1}{p_i^m} \\
&\quad = \sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{p_i^m} + \sum_{i=1}^{n} z_i \log_2 \frac{z_i}{u_i} - \sum_{i=1}^{n} z_i \log_2 \frac{z_i}{p_i^m}.
\end{aligned}$$

Hier ist

$$\sum_{j=1}^{m-1} \lambda^j \sum_{i=1}^{n} p_i^j \log_2 \frac{p_i^j}{p_i^m} + \sum_{i=1}^{n} z_i \log_2 \frac{z_i}{u_i} - \sum_{i=1}^{n} z_i \log_2 \frac{z_i}{p_i^m}$$

minimal, wenn

$$\sum_{i=1}^{n} z_i \log_2 \frac{z_i}{u_i} = 0.$$

Nach Satz 2.4.1 also genau dann, wenn $z_i = u_i \ \forall \ i \in \{1, \ldots, n\}$. Daraus folgt $\mu^j = \lambda^j \ \ \forall \ j \in \{1, \ldots, m\}$.

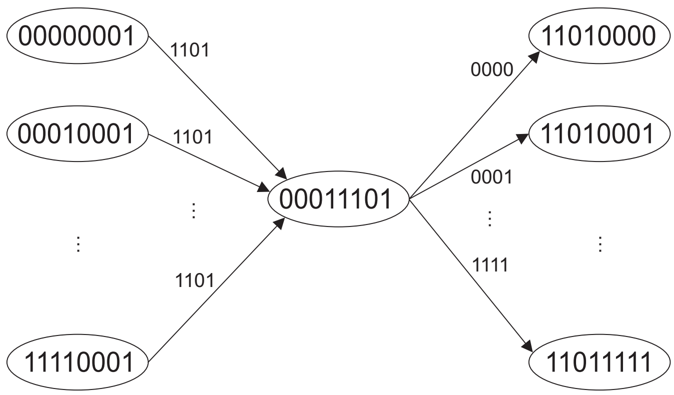
□

In diesem Abschnitt haben wir gesehen, wie Wahrscheinlichkeitsverteilungen effizient zusammengefügt werden können. Diese Konsolidierungsmethode wollen wir im Folgenden benutzen, um die Modellgrösse des Nibble-Modells mit Markowscher Kette zweiter Ordnung zu reduzieren.

Zunächst wollen wir uns allerdings das Zusammenfassen von Wahrscheinlichkeitsverteilungen am Markow-Graphen veranschaulichen.

## 4.2 Knotenkonsolidierung im Nibble-Modell

Für das Nibble-Modell mit Markowscher Kette zweiter Ordnung seien hier noch einmal Ausschnitte des zugehörigen Markow-Graphen für die Zustände 00011101 und 10001100 skizziert.



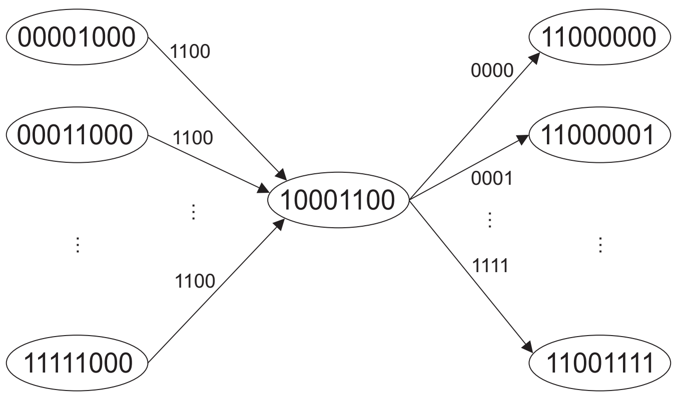



Abbildung 19: Nibble-Modell - Zustände 00011101 und 10001100

Eine Konsolidierung dieser beiden Knoten hat folgende Auswirkung:

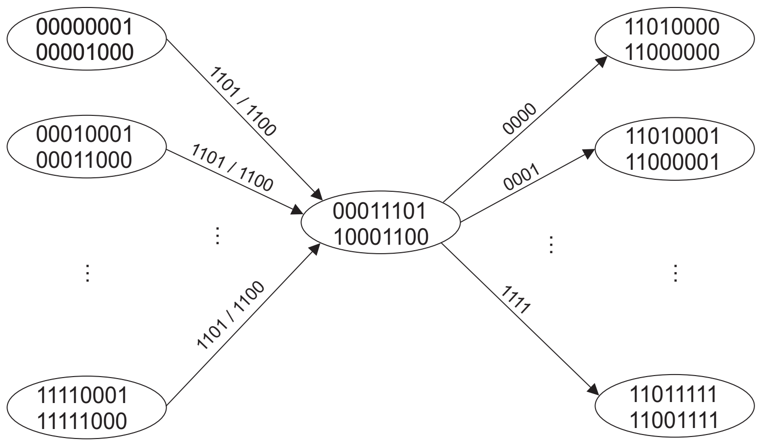



Abbildung 20: Konsolidierung der Zustände 00011101 und 10001100

Die beiden Zustände 00011101 und 10001100 werden zu einem Zustand zusammengefügt. Die beiden Zustände hatten zuvor in der Summe 32 ausgehende Kanten. Nach der Konsolidierung existieren nur noch derer 16. Wir haben also das Modell um genau eine Wahrscheinlichkeitsverteilung reduziert.

Um nun für einen praktischen Einsatz dieses Nibble-Modells die grosse Anzahl an Fliesskommazahlen zu reduzieren, benötigen wir konkrete Algorithmenideen, die die Knoten dieses Modells zusammenfügen.

## 4.3 Algorithmen und Testergebnisse

In diesem Abschnitt wollen wir nun drei verschiedene Algorithmen zur Knotenkonsolidierung im Nibble-Modell mit Markowscher Kette zweiter Ordnung betrachten.

### 4.3.1 Erster Algorithmus

Zur Beschreibung des folgenden Algorithmus sei P die Menge der Wahrscheinlichkeitsverteilungen des Nibble-Modells. Wir verwenden hier eine Pseudocode-Notation um die Arbeitsweise des Algorithmus zu erläutern.

**Algorithmus:**

```
for ( i=1; i < 4; i++ )
{
     Initialisiere P';
     while ( |P| > 0 )
     {
          Distanz = 1000;
          {
               Wähle ein p aus P;
               Für alle r aus P\{p}
               {
                    Berechne D(p||r) von p zu r;
                    if ( D(p||r) < Distanz )
                    {
                         Distanz = D(p||r);
                         q = r;
                    }
               }
               Berechne u aus p und q nach Satz (4.1.1);
               Füge u der Menge P' hinzu;
               Entferne p und q aus P;
          }
     }
     P = P';
}
```

Dieser Algorithmus halbiert pro Durchlauf der for-Schleife die Anzahl der Wahrscheinlichkeitsverteilungen. Mit drei Iterationen kommen wir somit auf 32 Verteilungen mit je 16 Elementen.

In der while-Schleife werden pro Durchlauf immer genau zwei Wahrscheinlichkeitsverteilungen zusammengefügt. Dazu wählt man zu Beginn eine Verteilung aus, und berechnet die Kullback Leibler Distanz zu allen anderen Verteilungen. Die Wahrscheinlichkeitsverteilung, zu der die gewählte Verteilung die geringste Distanz hat, wird herausgefiltert.

Anschliessend berechnet man nach Satz 4.1.1 aus den beiden Verteilungen eine Approximationsverteilung. Diese wird der Menge $P'$ hinzugefügt. Danach müssen die Verteilungen $p$ und $q$ noch aus der Menge $P$ entfernt werden. Nun startet die while-Schleife einen weiteren Durchlauf und der Vorgang wiederholt sich.

Die while-Schleife terminiert, nachdem die beiden letzten Wahrscheinlichkeitsverteilungen konsolidiert sind. Am Ende des Algorithmus befinden sich die 32 Approximationsverteilungen, aus denen das Modell nun besteht, in der Menge $P$.

Auf diese Weise haben wir aus 256 Verteilungen 32 errechnet. Dieses entspricht einer Modellreduktion auf 12.5 % der ursprünglichen Grösse. Im praktischen Fall müssen nun 512 Fliesskommazahlen für die Codierphase abgespeichert werden.

Da wir aber mit dieser Methode die tatsächlichen Wahrscheinlichkeitsverteilungen nur approximieren, entstehen für die Entropie folglich Verluste[18]. Die erwartete Codelänge wird sich also deutlich verschlechtern.

Um diese Verluste genau zu bestimmen, muss man für jede Verteilung die Kullback Leibler Distanz zur jeweiligen Approximationsverteilung ausrechnen. Der exakte Verlust pro Verteilung ergibt sich dann aus dem Resultat der Kullback Leibler Distanz, gewichtet mit der relativen Häufigkeit der Verteilung.

Die relative Häufigkeit einer Verteilung ist in diesem Fall jetzt die Anzahl, wie oft der Markowprozess in dem entsprechenden Zustand war, dividiert durch die Anzahl der Ereignisse des Prozesses.

Summiert man die Verluste für jede Wahrscheinlichkeitsverteilung des Modells auf, und addiert sie anschliessend zur Entropie der Markowschen Kette hinzu, so erhält man eine neue Abschätzung für die erwartete Codelänge.

Schauen wir uns die Resultate an, die die Implementierung dieses Algorithmus hervorgebracht hat.

| **Testergebnisse** | | | |
|---|---|---|---|
| Dateiname | Ordnung der Markow-Kette | Entropie pro Bit ohne Knotenkonsolidierung (256 Knoten) | Entropie pro Bit mit Knotenkonsolidierung (32 Knoten) |
| Application.bin | 2 | 0.6979 | 0.7909 |
| Undo.bin | 2 | 0.5291 | 0.6834 |
| TextParagraphSupplier.bin | 2 | 0.6048 | 0.7259 |
| Cut.bin | 2 | 0.5296 | 0.6967 |
| PictureChooser.bin | 2 | 0.6789 | 0.7775 |

Tabelle 4: Testergebnisse Modellreduktion - Erster Algorithmus

[18]Siehe auch das Beispiel zur Kullback Leibler Distanz in Abschnitt 2.4.

Die Ergebnisse zeigen, dass wir mit diesem Algorithmus mindestens 10 % Entropieverlust verursachen. Bei den Dateien 'Undo.bin' und 'Cut.bin' sind es sogar 16 %. Die Grösse des Modells ist zwar jetzt praktikabel, jedoch ist die erwartete Codelänge zu stark angestiegen.

Betrachtet man sich diesen Algorithmus etwas genauer, so stellen sich zwei wesentliche Nachteile heraus.

Einer davon ist das Filtern der Knoten mittels der Kullback Leibler Distanz. Es werden zwar mit dieser Methode Verteilungen zusammengefasst, die in einem gewissen Masse ähnlich sind, jedoch ist die Kullback Leibler Distanz nicht symmetrisch, siehe Abschnitt 2.4. Das bedeutet, dass wir für eine gewählte Verteilung $p$ eine Verteilung $r$ herausfiltern, zu der $p$ zwar die kürzeste Distanz besitzt, was jedoch nicht heisst, das auch $r$ die kürzeste Distanz zu $p$ besitzt.

Würde man also die Verteilung $r$ auswählen und die Kullback Leibler Distanzen zu allen anderen Wahrscheinlichkeitsverteilungen berechnen, so ist nicht gewährleistet, dass die Verteilung $p$ den geringsten Wert liefern würde.

Auf diese Weise werden also Verteilungen zusammengefasst, die nicht gegenseitig optimal zusammenpassen. Das wiederum bedeutet unnötigen Entropieverlust.

Der zweite Nachteil ist der Ablauf des Algorithmus. Pro Durchlauf der while-Schleife erzwingt das Verfahren die Konsolidierung zweier Verteilungen. Für die ersten Durchläufe ist das zunächst noch unproblematisch, jedoch werden mit zunehmender Iteration der while-Schleife immer zwei Knoten zusammengefügt, die schlecht zusammenpassen.

In der graphischen Darstellung des Modells bedeutet das, dass dieser Algorithmus einen Knoten zwingt, sich mit einem anderen Knoten zu verschmelzen. Auf diese Weise werden Knoten konsolidiert, die nicht gut zusammmenpassen und somit unnötigen Entropieverlust erzeugen.

Den ersten Nachteil, also das Filtern mittels Kullback Leibler Distanz, können wir mit dem folgenden Algorithmus eleminieren.

### 4.3.2 Zweiter Algorithmus

Wir ersetzen die Berechnung der Kullback Leibler Distanz von einer Wahrscheinlichkeitsverteilung $p$ zu einer Verteilung $r$ durch die Konstruktion der Approximationsverteilung $u$ von $p$ und $r$ nach Satz 4.1.1.

Mit Hilfe dieser Approximationsverteilung $u$ berechnen wir dann die gewichtete Kullback Leibler Distanz von $p$ und $r$ zu $u$. Wir betrachten also als Filtermethode den genauen Entropieverlust, den die Konsolidierung hervorruft.

Auf diesem Weg berücksichtigen wir bereits bei der Auswahl der Verteilungen, die zusammengefasst werden, die resultierende, zusätzlich zu erwartende Codelänge.

Zur Beschreibung des folgenden Verfahrens sei $a(p)$ die relative Häufigkeit der Verteilung p, die man erhält, indem man die Anzahl, wie oft der Markowprozess in dem entsprechenden Zustand war, durch die Anzahl der Ereignisse des Prozesses dividiert. Des weiteren sei $P$ wiederum die Menge der Wahrscheinlichkeitsverteilungen des Nibble-Modells mit Markowscher Kette zweiter Ordnung.

**Algorithmus:**

```
for ( i=1; i < 4; i++ )
{
     Initialisiere P';
     while ( |P| > 0 )
     {
          Distanz = 1000;
          {
               Wähle ein p aus P;
               Für alle r aus P\{p}
               {
                    Berechne u aus p und r nach Satz (4.1.1);
                    Berechne D(p||u) und D(r||u);
                    Verlust = a(p) * D(p||u) +  a(r) * D(r||u);
                    if ( Verlust < Distanz )
                    {
                         Distanz = Verlust;
                         q = r;
                    }
               }
               Berechne u aus p und q nach Satz (4.1.1);
               Füge u der Menge P' hinzu;
               Entferne p und q aus P;
          }
     }
     P = P';
}
```

Wir sehen, dass dieser Algorithmus, bis auf die beschriebenen Änderungen, identisch mit dem ersten Algorithmus ist.

Auch in dieser Version wird die Anzahl der Wahrscheinlichkeitsverteilungen pro Durchlauf der for-Schleife halbiert. Nach drei Iterationen erhalten wir demzufolge 32 Approximationsverteilungen.

Wie schon für den ersten Algorithmus, muss der Entropieverlust pro Verteilung ausgerechnet und der Entropie der Markowschen Kette hinzu addiert werden. Daraus resultiert dann wiederum die zu erwartende Codelänge.

Schauen wir uns die Testergebnisse dieses Verfahrens in Tabelle 5 an.

| Testergebnisse | | | |
|---|---|---|---|
| Dateiname | Ordnung der Markow-Kette | Entropie pro Bit ohne Knotenkonsolidierung (256 Knoten) | Entropie pro Bit mit Knotenkonsolidierung (32 Knoten) |
| Application.bin | 2 | 0.6979 | 0.7643 |
| Undo.bin | 2 | 0.5291 | 0.6204 |
| TextParagraphSupplier.bin | 2 | 0.6048 | 0.6766 |
| Cut.bin | 2 | 0.5296 | 0.6159 |
| PictureChooser.bin | 2 | 0.6789 | 0.7444 |

Tabelle 5: Testergebnisse Modellreduktion - Zweiter Algorithmus

Wir sehen, dass sich die Entropie nach der Knotenkonsolidierung gegenüber dem ersten Algorithmus wesentlich verbessert hat. Der durchschnittliche Verlust liegt in diesem Fall bei $\approx 8$ %.

Die Veränderung des Algorithmus verringert die zu erwartende Codelänge also deutlich. Wie schon erwähnt, begründet sich diese Verbesserung durch die Eigenschaft, dass für das Herausfiltern der zu konsolidierenden Verteilungen der genaue Entropieverlust der entsprechenden Zusammenfassung betrachtet wird. Bei einem Durchlauf der while-Schleife wird folglich zu einer gewählten Verteilung $p$ eine Verteilung $r$ gesucht, so dass die Summe der gewichteten Kullback Leibler Distanzen von $p$ und $r$ zur jeweiligen Approximationsverteilung $u$ minimal ist.

Wie die Testergebnisse zeigen, könnte man mit dieser Implementierung prinzipiell schon zufrieden sein. Eine Modellreduktion auf 12.5 % der ursprünglichen Modellgrösse bei einem gleichzeitigen Entropieverlust von circa 8 % ist ein akzeptables Resultat.

Bleibt die Frage, ob man dieses Ergebnis weiter optimieren kann. Wie schon beim ersten Algorithmus beschrieben, wird pro Iteration der for-Schleife die Anzahl der Wahrscheinlichkeitsverteilungen halbiert. Mit diesem Verfahren wird eine Verteilung dazu gezwungen, sich mit einer anderen Verteilung zu konsolidieren.

Mit anderen Worten, der Algorithmus nimmt wenig Rücksicht darauf, ob eine Wahrscheinlichkeitsverteilung wirklich gut zu einer anderen Wahrscheinlichkeitsverteilung passt. Spätestens mit der letzten Iteration würde eine Verteilung mit einer anderen Verteilung zusammengefasst, egal welchen Entropieverlust die Konsolidierung hervorruft.

In der graphischen Repräsentation heisst das wiederum, das ein Knoten früher oder später zum „heiraten“ gezwungen wird. Genau diese Eigenschaft ist die Schwachstelle des Algorithmus. Der im folgenden Abschnitt vorgestellte Algorithmus wird diese Problematik umgehen.

Diese ist allerdings nicht so einfach zu eleminieren, wie die Veränderung des zweiten Algorithmus zum Ersten.

### 4.3.3 Optimaler Algorithmus

Zur Beschreibung des optimalen Knotenkonsolidierungsalgorithmus sei P[] das Array, das alle Wahrscheinlichkeitsverteilungen des Nibble-Modells enthält. P[$i$] sei also die $i$-te Verteilung, $i \in \{0, \ldots, 255\}$. Der Algorithmus arbeitet wie folgt.

**Algorithmus:**

```
for ( i=0; i < 224; i++ )
{
    Iteration = |P[]|;
    Initialisiere A[][];
    for ( p=0; p < Iteration; p++ )
    {
         for ( q=0; q < Iteration; q++ )
         {
              if ( p != q )
              {
                   Berechne u aus P[p] und P[q] nach (4.1.1);
                   Berechne D(P[p]||u) und D(P[q]||u);
                   Verlustp = a(P[p]) * D(P[p]||u);
                   Verlustq = a(P[q]) * D(P[q]||u);
                   A[p][q] = Verlustp + Verlustq;
              }
         }
    }
    Minimum = 1000;
    for ( p=0; p < Iteration; p++ )
    {
         for ( q=0; q < Iteration; q++ )
         {
              if ( p != q )
              {
                   Min = A[p][q];
                   if ( Min < Minimum )
                   {
                        Minimum = Min;
                        VA = p;
                        VB = q;
                   }
              }
         }
    }
    Berechne u aus P[VA] und P[VB] nach Satz (4.1.1);
```

```
    Füge u dem Array P[] hinzu;
    Entferne P[VA] und P[VB] aus P[];
    }
}
```

In jedem Durchlauf der for-Schleife wird die Anzahl der Wahrscheinlichkeitsverteilungen um genau eine Verteilung reduziert. Um wie in den vorherigen Algorithmen auf 32 Verteilungen, also auf 512 Fliesskommazahlen, zu reduzieren, läuft die äussere for-Schleife hier 224-mal.

Die erste doppelte for-Schleife berechnet eine Matrix A, die als Einträge A[$i$][$j$] die Summe der gewichteten Kullback Leibler Distanzen von der Verteilung $i$ und der Verteilung $j$ zur, aus $i$ und $j$ berechneten, Approximationsverteilung $u$ enthält.

Nach dem Durchlauf der beiden Schleifen beinhaltet diese Matrix an der Position A[$i$][$j$] also den zu erwartenden Entropieverlust, wenn man die Verteilung $i$ und die Verteilung $j$, $1 \leq i, j \leq |P|$, nach Satz 4.1.1 konsolidiert.

Die Position A[$i$][$j$], $1 \leq i, j \leq |P|$, die den kleinsten Wert der gesamten Matrix enthält, repräsentiert nun die beiden Verteilungen, die bei einer Konsolidierung den geringsten Entropieverlust verursachen würden. Die Zeile $i$ entspricht dabei der Verteilung P[$i$] und die Spalte $j$ der Verteilung P[$j$].

Die zweite doppelte for-Schleife des obigen Algorithmus findet die genaue Position des kleinsten Eintrags in A. Zudem speichert sie die genaue Zeile und Spalte in den Variablen VA und VB.

Nachdem diese Schleifen durchlaufen sind, muss die Approximationsverteilung $u$ aus den Verteilungen P[$VA$] und P[$VB$] gemäss Satz 4.1.1 berechnet werden. Diese wird dann wiederum der Menge der Wahrscheinlichkeitsverteilungen P[] hinzugefügt. Da P[$u$] nun die Verteilungen P[$VA$] und P[$VB$] approximiert, müssen wir P[$VA$] und P[$VB$] noch aus P[] entfernen.

Auf diese Weise haben wir in einer Iteration die Anzahl der Wahrscheinlichkeitsverteilungen um genau eine Verteilung reduziert, und hierbei den Entropieverlust minimiert.

Des weiteren wird mit diesem Algorithmus keine Wahrscheinlichkeitsverteilung gezwungen, sich mit einer anderen Verteilung zu konsolidieren.

Dieses Verfahren berücksichtigt also den Fall, dass sofern eine Verteilung absolut zu keiner anderen Verteilung passt, diese dann auch nicht konsolidiert wird. Vielmehr erlaubt es die äussere for-Schleife, die gewünschte Anzahl an Verteilungen zu bestimmen. Ist der Wert der Entropie mit 32 Verteilungen zu schlecht, so kann man den Parameter der for-Schleife problemlos ändern, und beispielsweise 40 statt 32 Verteilungen verwenden.

Bei den beiden vorherigen Algorithmen ist eine Parametereinstellung dieser Art nicht möglich. Ist die Entropie bei 32 Verteilungen dort zu schlecht, so wäre die nächste Stufe eine Approximation mit 64 Verteilungen. Natürlich ist die Entropie dann etwas besser, jedoch hat man sofort den doppelten Speicheraufwand.

Die Implementierung des optimalen Konsolidierungsalgorithmus hat die folgenden Ergebnisse geliefert. Um die Verfahren vergleichen zu können, reduzieren wir auch hier auf 32 Wahrscheinlichkeitsverteilungen.

| **Testergebnisse** | | | |
|---|---|---|---|
| Dateiname | Ordnung der Markow-Kette | Entropie pro Bit ohne Knotenkonsolidierung (256 Knoten) | Entropie pro Bit mit Knotenkonsolidierung (32 Knoten) |
| Application.bin | 2 | 0.6979 | 0.7445 |
| Undo.bin | 2 | 0.5291 | 0.5812 |
| TextParagraphSupplier.bin | 2 | 0.6048 | 0.6531 |
| Cut.bin | 2 | 0.5296 | 0.5801 |
| PictureChooser.bin | 2 | 0.6789 | 0.7247 |

Tabelle 6: Testergebnisse Modellreduktion - Optimaler Algorithmus

Wie wir der Tabelle 6 entnehmen können, hat sich der Wert der Entropie nach der Knotenkonsolidierung noch einmal wesentlich verbessert. Mit dem in diesem Abschnitt vorgestellten Konsolidierungsalgorithmus liegt der Entropieverlust bei $\approx 5$ %.

Wir sind also in der Lage, das Nibble-Modell mit Markowscher Kette zweiter Ordnung auf 12.5 % seiner Grösse zu reduzieren und gleichzeitig einen Entropieverlust von lediglich $\approx 5$ % zu erzeugen. Mit 512 zu speichernden Fliesskommazahlen wird dieses Modell praktikabel und verliert nicht zu viel seiner guten erwarteten Kompression.

Bei genauer Analyse dieses Verfahrens hat sich herausgestellt, dass einige Wahrscheinlichkeitsverteilungen gar nicht konsolidiert werden. Diese Beobachtung entspricht genau dem Nachteil der anderen Verfahren, die die Konsolidierung erzwungen haben. Das Problem der vorherigen Algorithmen, das in den letzten Durchläufen der while-Schleife Verteilungen zusammengefasst werden, die nicht richtig zusammenpassen, wird bei diesem Algorithmus eleminiert.

Des weiteren hat sich herausgestellt, dass eine einzige Approximationsverteilung bis zu 30 Original-Verteilungen beinhalten kann. Andere Verteilungen approximieren wiederum nur zwei oder drei Wahrscheinlichkeitsverteilungen des ursprünglichen Modells. Natürlich hängt das genaue Konsolidierungsverhalten von der entsprechenden Datei ab, die analysiert wird.

Die Arbeitsweise dieses Algorithmus ist sehr viel präziser als die Arbeitsweise der vorherigen Verfahren. Bei jeder Iteration wird ausschliesslich optimal konsolidiert. Der resultierende Gesamtverlust ist somit der Verlust, der unumgänglich ist. Mit anderen Worten, um den Verlust zu reduzieren, muss die Knotenanzahl erhöht, also die Anzahl der Iterationen verkleinert werden.

Einen kleinen Nachteil hat dieser Algorithmus allerdings, denn seine Rechenzeit ist deutlich höher als die Rechenzeit der vorherigen Verfahren. Da eine Implementierung dieses Algorithmus, gerade für den Einsatz in eingebetteten Systemen, ohnehin in Hardware umgesetzt würde, sind Laufzeitanalysen an dieser Stelle überflüssig.

In diesem Kapitel haben wir nun verschiedene Techniken zur Modellreduktion des Nibble-Modells mit Markowscher Kette zweiter Ordnung kennengelernt. Hervorhebend ist hierbei natürlich der optimale Algorithmus aus Abschnitt 4.3.3.

Bei einer Modellreduktion auf 12.5 % der Originalgrösse resultiert ein Entropieverlust von ungefähr 5 %. Somit haben wir ein von der Speicherkomplexität her praktikables Modell, das im Durchschnitt eine gute Kompression erwarten lässt.

# 5 Fazit

Wir haben in dieser Arbeit verschiedene Modellierungsmöglichkeiten kennengelernt, um unter Verwendung von stochastischen Prozessen die Binärdarstellung ausführbarer Computerprogramme, in diesem Fall JAVA-Bytecode, für eine Kompression zu charakterisieren.

Hierbei haben sowohl die Bit- als auch die 2-Bit-Modelle, wie sie hier beschrieben sind, keine gute Modellierung hervorgebracht. Anders hingegen das Nibble-Modell mit Markowscher Kette zweiter Ordnung. Die Kompression, die diese Modellierung erwarten lässt, ist überdurchschnittlich gut.

Da die Komplexität dieses Modells für einen praktischen Einsatz aber deutlich zu gross ist, haben wir verschiedene Algorithmen betrachtet, die die Modellgrösse reduzieren. Besonders gute Ergebnisse erzielte dabei das in Abschnitt 4.3.3 vorgestellte Verfahren.

Zusammen mit diesem Knotenkonsolidierungsalgorithmus liefert uns das Nibble-Modell mit Markowscher Kette zweiter Ordnung eine gute Modellbildung, mit der beispielsweise die arithmetische Codierung oder auch die Huffman-Codierung in der zweiten Phase des semi-adaptiven Modells codieren kann.

Die erwartete Codekompression auf $\approx 60 - 65\,\%$ der Originalgrösse entspricht in etwa der Kompression, die Haris Lekatsas und Wayne Wolf mit ihren positionsabhängigen Bit-Modellen, bei ähnlicher Modellgrösse, in der Arbeit „SAMC: A Code Compression Algorithm for Embedded Processors“ [LW99b] erreicht haben. Teilweise sind die Ergebnisse von Lekatsas und Wolf besser, da in deren Arbeit Programm-Code von Instruktionssätzen getestet wurde, der viel Redundanz beinhaltet.

Grundsätzlich kann man die Ergebnisse aber nicht vergleichen, da die Modelle in dieser Arbeit mit völlig anderen Eingaben getestet wurden, als die Modelle bei Lekatsas und Wolf. Es sei hier lediglich erwähnt, dass die erwartete Kompression, die mit unserer Modellierung erreicht wird, in etwa dem entspricht, was ähnliche Markow-Modelle auf anderen Instruktionssätzen erreichen.

In einem eingebetteten System, zum Beispiel der Codekompression auf Smartcards, bei dem es auch auf die Codier- und Decodierlaufzeit ankommt, muss sicherlich gezielt überlegt werden, ob ein semi-adaptives Modell überhaupt realisierbar ist.

Wahrscheinlicher ist in dem Fall, dass ein festes Modell mit festen Wahrscheinlichkeitsverteilungen, für alle gegebenen Programme eines bestimmten Instruktionssatzes, verwendet wird. Um dieses zu realisieren, ist die Analyse einer grösseren Anzahl von Programmen notwendig.

Die erste Phase des semi-adaptiven Modells muss also mit verschiedenen Programmen des gleichen Instruktionssatzes vorab durchgeführt werden. Aus den resultierenden Ergebnissen kann dann ein Modell zusammengefasst werden, mit dem das Codierverfahren arbeitet.

Man könnte an dieser Arbeit ansetzen, indem man beispielsweise die tabellenbasierte arithmetische Codierung von Howard und Vitter auf diese Modellierung anwendet, und somit Computerprogramme komprimiert.

# Literatur


[CH87] G.V. Cormack and R.N. Horspool. Data compression using dynamic markov modelling. *The Computer Journal*, 30(6), 1987.

[CT91] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., New York, 1991.

[GS01] G.R. Grimmett and D.R. Stirzaker. *Probability and Random Processes*. Oxford University Press, New York, 2001.

[Kre00] U. Krengel. *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Vieweg, Braunschweig/Wiesbaden, 2000.

[LW98] H. Lekatsas and W. Wolf. Code compression for embedded systems. *Proceedings of the Design Automation Conference*, Juni 1998.

[LW99a] H. Lekatsas and W. Wolf. Random access decompression using binary arithmetic coding. *Proceedings of the 1999 IEEE Data Compression Conference*, März 1999.

[LW99b] H. Lekatsas and W. Wolf. SAMC: A code compression algorithm for embedded systems. *IEEE Transactions on CAD*, 18(12), 1999.

[PJ92] P.G.Howard and J.S.Vitter. Practical implementations of arithmetic coding. *Image and Text Compression, Kluwer Academic Publishers, Norwell, MA*, pages 85–112, 1992.

[Say00] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers, San Francisco, 2000.

# Index


2-Bit-Modelle, 33, 41, 62

Alphabet, 3, 5–8, 11–13

bedingte Entropie, 15
bedingte Wahrscheinlichkeit, 14
Bit-Modelle, 20, 31, 33, 40, 62

Codekompression, 12, 18, 19, 31, 33, 41, 45, 46, 62
Codierung, 3–5, 25
    arithmetische, 13, 18, 19, 24, 47, 62
    Huffman, 13, 18, 19, 24, 62

Datenkompression, 3, 5, 12, 13, 18, 19, 33

eindeutig entschlüsselbar, 3–5
Entropie, 6–9, 11, 12, 22, 23, 31, 33, 35, 36, 40, 41, 46, 54–56
erinnerungslose Quelle, 3
erwartete Länge eines Codes, 5, 8, 16, 17

kompakt, 5
Kompression, 5, 6, 13, 31, 36, 41
Kraft, 5, 8
Kullback Leibler Distanz, 9–11, 48, 53–55

Markow-Graph, 22, 27, 28, 34, 44
Markow-Modell, 19
Markowprozesse, 13, 19, 25
Markowsche Kette, 1, 14
    dritter Ordnung, 27
    erster Ordnung, 15, 20, 22, 23, 33, 39, 41, 44, 46
    k-ter Ordnung, 15, 29, 45
    zweiter Ordnung, 15, 25, 26, 33, 36, 43, 44, 46, 47, 51, 53, 56, 60, 62
McMillan, 5–7

Nachricht, 3, 4, 11, 12, 19–23, 26–31, 33–36, 39, 41, 42, 44, 46
Nibble, 41–44
Nibble-Modelle, 41, 45, 47, 51, 58, 60, 62

Präfix-Code, 4

Quelle, 3, 5, 7–9, 11, 13, 19
    erinnerungslose, 3, 13, 18

semi-adaptives Modell, 19, 24, 62
stochastischer Prozess, 14

Wahrscheinlichkeitsraum, 13
Wahrscheinlichkeitsverteilung, 1, 3, 5–9, 12, 13, 21, 22, 24, 33, 37, 39, 42, 44, 47–51, 54, 55, 57

**Erklärung**

Ich versichere, dass ich diese Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe.

Paderborn, den 5. November 2003

Ulrich Hoppe
Lange Wiese 40
34439 Willebadessen