Datenkompression

Maciej Liśkiewicz, Henning Fernau
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen
liskiewi,fernau@informatik.uni-tuebingen.de

Inhaltsverzeichnis

Vorwort

Dieses Scriptum entstand als Begleitmaterial zur zweistündigen Vorlesung "Datenkompression" im Sommersemester 1999 an der Universität Tübingen, Wilhelm-Schickard-Institut für Informatik.

Wesentliche Grundlage des Scriptums ist das Buch von Sayood [10], auf das im laufenden Scripttext immer wieder verwiesen wird.

Danken wollen wir an dieser Stelle der guten und kritischen Mitarbeit der studentischen Hörer der Vorlesung. Wir möchten Sie, liebe Leser, auffordern, uns auch weiterhin mit Ihrer Kritik bei der Verbesserung des Scriptums zu helfen. Ferner danken wir Mark Dokoupil für seine Mühen und Hilfen beim Erstellen von Bildern, Tabellen und Folien mit xfig und LaTeX sowie für die sprachliche Bearbeitung.

1 Einleitung

Unter Datenkompression verstehen wir zwei Algorithmen: Der erste Algorithmus konstruiert für die Eingabe $\mathcal{X}$ eine Repräsentation $\mathcal{X}_c$, die weniger Bits als $\mathcal{X}$ braucht, und der zweite Algorithmus generiert für gegebenes $\mathcal{X}_c$ die Rekonstruktion $\mathcal{Y}$. Ein Datenkompressionschema heißt verlustfrei, wenn $\mathcal{X} = \mathcal{Y}$; sonst nennen wir das Schema Lossy Komprimierung.

Ein natürliches Maß für die Qualität des Komprimierungsschemas ist der Quotient: "Anzahl der Bits von $\mathcal{X}$" durch "Anzahl der Bits von $\mathcal{X}_c$", den wir *Kompressionsquotient* (engl.: compression ratio) nennen. In der Praxis vergleichen wir die Komprimierungsschemata bezüglich des Kompressionsquotienten so, daß bestimmte Eingaben $\mathcal{X}$ betrachtet werden. Zum Beispiel, um die Methoden der Komprimierung von Bildern zu vergleichen, benutzt man acht bestimmte schwarz-weiß Testbilder, *CCITT fax test images* genannt. Die Tabelle unten zeigt den globalen Kompressionsquotienten für verschiedene lossless Methoden, das heißt die Größe aller Bildern durch die gesamte Größe der komprimierten Repräsentationen (mehr über *CCITT fax test* finden Sie z.B. in
`http://www.cs.waikato.ac.nz/ singlis/ratios.html`).

Jahr	Schema	Quotient
1980	CCITT Group 3, T.4	7.7
1984	CCITT Group 4, T.6	15.5
1988	IBM's Q-Coder	19.0
1991	2-level coding	21.4
1993	JBIG	19.7
1995	TIC	22.3

Ein weiteres Maß für die Qualität ist die durchschnittliche Anzahl von Bits für die Repräsentation eines einzigen Samples (z.B. Pixel, Symbol).

Für verlustbehaftete Komprimierung braucht man noch ein Maß, um die Qualität der Rekonstruktion $\mathcal{Y}$ zu bewerten. Den Unterschied zwischen $\mathcal{Y}$ und $\mathcal{X}$ nennen wir die *Entstellung* (engl.: distortion) und den Begriff werden wir in Kapitel 6 diskutieren.

1.1 Ein bißchen Informationstheorie

Betrachten wir das folgende Problem: Wie lautet eine sinnvolle Definition für das quantitative Maß der Information? Elwood Shannon hat das Maß wie folgt definiert: Sei A ein Ereignis und $P(A)$ die Wahrscheinlichkeit, daß A eintritt

$$i(A) = \log_{(2)} \frac{1}{P(A)}.$$

Shannon hat $i(A)$ Informationsgehalt genannt. Intuitiv ist der Begriff klar: Seien A und B zwei Ereignisse (z.B. $A =$ "Der Verbrecher ist kleiner als 2m." und $B =$ "Der Verbrecher ist größer als 2m.") mit den Wahrscheinlichkeiten

$$P(A) = 0.999 \quad \text{und} \quad P(B) = 0.001.$$

Eine Nachricht von Zeugen über A ist keine große Überraschung und deshalb enthält sie nicht viel Information ($i(A)$ ist sehr klein). Eine Nachricht, daß B wahr ist, enthält dagegen viel Information ($i(B)$ ist sehr groß). Sei S eine bestimmte Informationsquelle. Die Entropie ist der "mittlere Informationsgehalt pro Nachricht aus S":

$$H(S) = \sum_{A \in S} P(A) \cdot i(A),$$

wobei wir für $P(A) = 0 \quad P(A) \cdot i(A)$ als 0 betrachten (dies ist sinnvoll, weil $\lim_{x \to 0} x \cdot \log \frac{1}{x} = 0$). Wir werden meistens S als endliches Quellenalphabet $\Sigma = \{a_1, \ldots, a_n\}$ betrachten. Denn $P(a_i)$ ist die Wahrscheinlichkeit, daß a_i enthalten ist und self-information $i(a_i) = \log_2 1/P(a_i)$ beschreibt die Anzahl von Bits (deshalb Basis 2) für die Kodierung a_i. Die Entropie

$$H(\Sigma) = \sum_{i=1}^{n} P(a_i) \cdot \log \frac{1}{P(a_i)}$$

ist die mittlere Anzahl von Bits, um die Message aus Σ zu kodieren.
Beispiele zur Entropie: Sei $\Sigma = \{a, b, c, d, e\}$ mit $P(a) = P(b) = P(c) = 0.25$ und $P(d) = P(e) = 0.125$. Dann ist

$$H = 3 \cdot 0.25 \cdot \log 4 + 2 \cdot 0.125 \cdot \log 8 = 1.5 + 0.75 = 2.25.$$

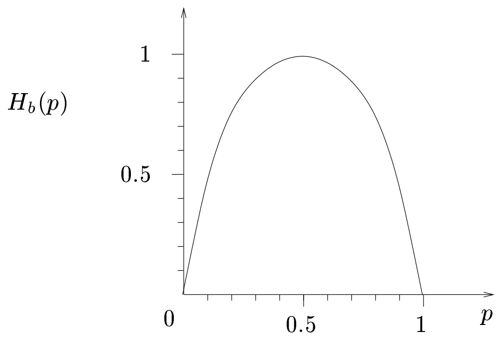
Weitere Beispiele:

	$P(a)$	$P(b)$	$P(c)$	$P(d)$	$P(e)$	H
1.	0.2	0.2	0.2	0.2	0.2	2.322
2.	0.5	0.25	0.125	0.0625	0.0625	1.875
3.	0.75	0.0625	0.0625	0.0625	0.0625	1.3
4.	0.94	0.01	0.01	0.01	0.01	0.322

Eine interessante Eigenschaft ist, daß die Entropie für ungleichmäßigere Verteilungen kleiner wird. Als weiteres Beispiel, das diese Eigenschaft illustriert, betrachten wir die binäre Entropie-Funktion:

$$H_b(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p},$$

das heißt, die Entropie-Funktion für zwei Quellensymbole mit Wahrscheinlichkeiten p und $p-1$. Die Werte für $H_b(p)$, wobei p zwischen 0 und 1 variiert, zeigt das Diagramm unten.



Weitere Informationen finden Sie z.B. in [13].

1.2 Benchmarks für die englische Sprache

Um ein Komprimierungsschema für Files mit Texten in Englisch abzuschätzen, muß man wissen, wieviel Information die englische Sprache besitzt.

Nehmen wir an, daß wir die Information in Bits per Zeichen messen und daß wir 96 druckfähige ASCII Zeichen benutzen. Dann bekommen wir für die uniforme Verteilung der Zeichen, daß die die Entropie gleich $\log 96 = 6.6$ Bits/Zeichen, also sehr groß ist. Wenn aber die Verteilung der englischen Sprache entspricht, die auf einer Sammlung englischer Texte basiert, dann bekommen wir für die Entropie etwa 4.5 Bits/Zeichen. Für die Codierung, bei der jedes Zeichen separat codiert wird, ist die Huffman-Methode optimal und in dem Fall ist 4.7 – die erwartete Anzahl von Bits pro Zeichen – nur ein bißchen schlechter als die Entropie.

Daß die Entropie 4.5 ist, heißt natürlich nicht, daß man englische Texte nicht besser komprimieren kann als mit 4.5 Bits pro Zeichen. Nehmen wir an, daß wir Blöcke von Zeichen der Länge 8 betrachten. Dann bekommen wir 96^8 verschiedene Blöcke und diese versuchen wir jetzt zu codieren. Eine interessante Beobachtung ist, daß das für die uniforme Verteilung nicht hilft:

Die Entropie für die neue Quelle ist jetzt $\log 96^8 = 8 \times 6.6$ Bits/Block = 6.6 Bits/Zeichen. Also genau so groß wie früher. Für die Verteilung, die der englischen Sprache entspricht, bekommen wir aber eine viel bessere untere Schranke für die Anzahl von Bits pro Zeichen; die Enropie ist hier nämlich etwa 19 Bits/Block und das heißt 2.4 Bits/Zeichen. Wenn wir die Länge der Blöcke weiter vergrößern, dann werden wir eine immer bessere Entropie bekommen. Wegen der enormen Anzahl von Blöcken mit großer Länge ist es leider unmöglich, eine präzise Statistik zu bekommen. Man vermutet, daß die untere Schranke für die Entropie etwa bei 1.3 Bits/Zeichen liegt.

In der Praxis entwickelt man Komprimierungsmethoden, die für englische Texte verschiedener Art effizient sind. Als Benchmark wird für diese Zwecke normalerweise der sogenannte *Calgary Corpus* verwendet. Diese Sammlung enthält 2 Bücher, 5 Artikel, 1 Literaturliste, 1 Sammlung von Zeitungsartikeln, 3 Programme, 1 terminal session, 2 binäre Programme, 1 geographische Datei und 1 Bit-Map von schwarz-weiß Bildern. Die Tabelle unten zeigt die Entwicklung der Komprimierungsmethoden durch die Jahren für den *Calgary Corpus* als Benchmark.

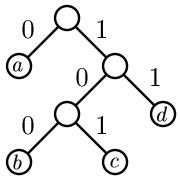
Datum	bpc	Schema	Autoren
May 1977	3.94	LZ77	Ziv, Lempel
1984	3.32	LZMW	Miller and Wegman
1987	3.30	LZH	Brent
1987	3.24	MTF	Moffat
1987	3.18	LZB	Bell
.	2.71	GZIP	.
1988	2.48	PPMC	Moffat
.	2.47	SAKDC	Williams
.	2.47	PPMD	Howard
Nov 1993	2.34	PPMC	Moffat
Oct 1994	2.34	PPM*	Cleary, Teahan, Witten
18 Nov 1994	2.33	PPMD	Moffat
1995	2.29	BW	Burrows, Wheeler
31 Jan 1995	2.27	PPMD	Teahan
1997	1.99	BOA	

2 Grundlegende Codes

Betrachten wir ein Codierungverfahren, das jedem Symbol aus dem Quellenalphabet genau ein binäres Wort zuordnet. Die Codierung hat zusätzlich folgende Eigenschaft: Kein Code ist ein Präfix des anderen Code. Die Codes, die diese Eigenschaft haben, nennen wir **Präfixcodes**. Mit Hilfe von Präfixcodes können wir eine Reihenfolge von Symbolen so codieren, daß wir einfach die Codes hintereinander schreiben, ohne zusätzliche Trennsymbole zu benutzen. In diesem Kapitel beschreiben wir den grundlegenden Präfixcode – den Huffman-Code. Dann vergleichen wir den Code mit einem weiteren Beispiel des Präfixcodes, nämlich mit der Shannon-Fano Codierung.

2.1 Huffman-Algorithmus

Im grundlegenden Huffman-Code ist jedes Symbol ein Blatt in einem Binärbaum und das Symbol wird durch einen Weg von der Wurzel bis zum Blatt codiert (links = 0, rechts = 1). Beispiel:



Für das Alphabet $\Sigma = \{a_1, ..., a_k\}$, wobei jedes Symbol a_i mit Wahrscheinlichkeit $P(a_i)$ auftritt, und die Codierung $K : \Sigma \to \{0,1\}^*$ definieren wir die erwartete Länge von K als

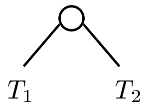
$$L_K := \sum_{i=1}^{k} P(a_i) \cdot |K(a_i)|.$$

Huffman-Codes (wir definieren sie weiter unten noch genau) sind optimale Präfixcodes (also Codes mit kleinster erwarteter Länge L_K).

Der Algorithmus für Huffman-Codierung.

1. `Starte mit dem Wald aus Bäumen, in dem jeder Baum ein Symbol darstellt und das Gewicht des Baumes ist` $w_i = P(a_i)$.
2. `Repeat until ein Baum im Wald`
   - `wähle die zwei Bäume` T_1 `und` T_2 `mit den kleinsten Gewichten` w_1 `und` w_2`;`

- nimm nun statt T_1 und T_2 den Baum

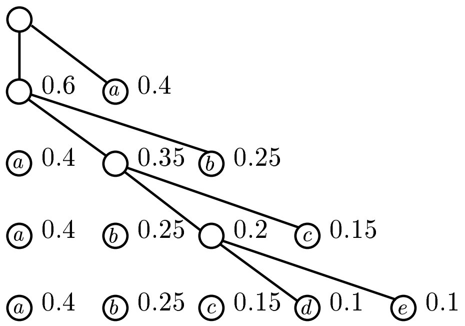


mit dem Gewicht $w_1 + w_2$.

Beispiel. Betrachten wir das Eingabealphabet $\Sigma = \{a, b, c, d, e\}$ und die Wahrscheinlichkeiten:

$P(a)$	$P(b)$	$P(c)$	$P(d)$	$P(e)$
0.4	0.25	0.15	0.1	0.1

Der Algorithmus konstruiert den Ausgabebaum folgendermaßen:



Der Ausgabecode ist:

a	b	c	d	e
0	11	101	1000	1001

Man kann induktiv das folgende Ergebnis beweisen.

Satz 2.1 *Die erwartete Codelänge für Huffman-Codierung ist optimal.*

Betrachten wir jetzt die Qualität der Huffman-Codierung in Vergleich mit der Entropie oder anders gesagt vergleichen wir die optimale erwartete Codelänge mit der Entropie.

Satz 2.2 *Sei Σ ein Quellenalphabet und $K : \Sigma \to \{0,1\}^*$ ein beliebiger Präfixcode. Dann gilt:*

$$L_K \geq H(\Sigma).$$

Satz 2.3 *Für jedes Quellenalphabet Σ ist die minimale erwartete Codelänge für Präfixcodes höchstens $H(\Sigma) + 1$.*

Dann bekommen wir als Korollar die folgende Abschätzung für die erwartete Länge der Huffman-Codierung:

Korollar 2.4 *Für jedes Quellenalphabet* Σ *gilt:*

$$H(\Sigma) \leq L_H \leq H(\Sigma) + 1.$$

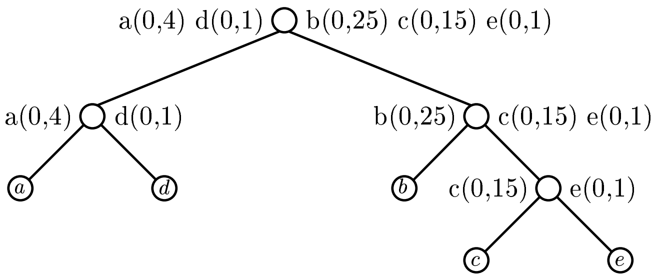
2.2 Shannon-Fano Codierung

Ein weiteres Beispiel für einen Präfixcode ist die Shannon-Fano Codierung:

make-code(Σ)
  if $|\Sigma| = 1$ then for $a \in \Sigma$ return node(a)
  else
    split Σ into Σ_1 and Σ_2 such that $\sum_{a\in\Sigma_1} P(a) \approx \sum_{a\in\Sigma_2} P(a)$;
    return node(make-code(Σ_1),make-code(Σ_2)).

Die Codierung zeigen wir nur als Beispiel. Im Vergleich mit der Huffman-Codierung ist die Methode von Shannon-Fano viel schlechter. Sie konstruiert z. B. nicht immer optimale Präfixcodes.

Beispiel. Betrachten wir die gleiche Eingabe wie für den (grundlegenden) Huffman-Algorithmus, das heißt $P(a) = 0.4,\ P(b) = 0.25,\ P(c) = 0.15$ und $P(d) = P(e) = 0.1$. Der Shannon-Fano Algorithmus arbeitet folgendermaßen



Die Ausgabecode ist:

	P	Code	Länge
a	0.4	00	2
b	0.25	10	2
c	0.15	110	3
d	0.1	01	2
e	0.1	111	3

Für diese Codierung ist die erwartete Länge

$$L_{S\text{-}F} = 2 \cdot 0.4 + 2 \cdot 0.25 + 3 \cdot 0.15 + 2 \cdot 0.1 + 3 \cdot 0.1 = 2.25,$$

wobei hingegen die (optimale) erwartete Länge für die Huffman-Codierung

$$L_H = 1 \cdot 0.4 + 2 \cdot 0.25 + 3 \cdot 0.15 + 2 \cdot 4 \cdot 0.1 = 2.15$$

ist.

2.3 Erweiterte Huffman-Codierung

Betrachten wir folgendes Beispiel: Sei $\Sigma = \{a, b\}$ und $P(a) = 0.9,\ P(b) = 0.1$. Dann gibt der Huffman-Algorithmus folgende Codierung:

	P	Code
a	0.9	0
b	0.1	1

mit der erwarteten Länge $1 \cdot 0.9 + 1 \cdot 0.1 = 1$ Bits/Symbol. Die Entropie für unser Alphabet (also die mittlere Anzahl von Bits, um das Symbol aus Σ zu kodieren) ist

$$H(\Sigma) = 0.9 \cdot \log \frac{10}{9} + 0.1 \cdot \log 10 = 0.47\ .$$

Die Redundanz – die Differenz zwischen der erwarteten Länge und der Entropie – ist 0.53 Bits/Symbol, also fast 113% der Entropie. Das heißt, daß (1) unser Code praktisch nicht komprimiert (er 'übersetzt' nur $a \to 0$ und $b \to 1$) und (2) um einen Text zu codieren, benutzen wir um 113% mehr Bits als eine minimale Codierung braucht. Um diesen Unterschied zu verkleinern, betrachten wir das neue Alphabet

$$\Sigma^2 = \{aa, ab, ba, bb\}$$

mit der Wahrscheinlichkeiten: $P(aa) = [P(a)]^2,\ P(bb) = [P(b)]^2$ und $P(ab) = P(ba) = P(a) \cdot P(b)$. Der Huffman-Algorithmus konstruiert folgende Codierung:

	P	Code	Länge
aa	0.81	0	1
ab	0.09	10	2
ba	0.09	110	3
bb	0.01	111	3

Die erwartete Länge ist $1 \cdot 0.81 + 2 \cdot 0.09 + 3 \cdot 0.09 + 3 \cdot 0.01 = 1.29$ Bits/Symbol in Σ^2 Alphabet, also 0.645 Bits/Symbol in Σ und jetzt benutzt unsere Codierung nur 37% mehr Bits als eine minimale Codierung. Für das Alphabet

$$\Sigma^3 = \{aaa, aab, aba, baa, abb, bab, bba, bbb\}$$

ist der Unterschied noch kleiner:

	P	Code	Länge
aaa	0.729	0	1
aab	0.081	100	3
aba	0.081	101	3
baa	0.081	110	3
abb	0.009	11100	5
bab	0.009	11101	5
bba	0.009	11110	5
bbb	0.001	11111	5

und die erwartete Länge ist 0.52 Bits/Symbol: nur um 11% mehr als die Entropie $H(\Sigma)$.

Sei H^m die erweiterte Huffman-Codierung für die Blöcke der Länge m. Dann bekommen wir die folgende Ungleichung.

Satz 2.5 *Für jedes Quellenalphabet Σ gilt es:*

$$H(\Sigma) \leq L_{H^m} \leq H(\Sigma) + \frac{1}{m}.$$

Beweis: Wir zeigen, daß $H(\Sigma^m) = mH(\Sigma)$ und dann benutzen wir Korollar 2.4. $\square$

Wir haben gesehen, daß die Wahrscheinlichkeiten für jedes neue "Zeichen" $a_{i_1}a_{i_2}\ldots a_{i_m}$ des Alphabets Σ^m das Produkt von Wahrscheinlichkeiten $P(a_1)\times P(a_2)\times\ldots P(a_m)$ ist. Das heißt, daß die Wahrscheinlichkeiten für die Zeichen innerhalb eines Blocks unabhängig sind. In der Praxis haben wir aber sehr selten mit einer solchen Situation zu tun. Für die englische Sprache z.B. ist die Wahrscheinlichkeit, daß das Zeichen "a" im U. S. Grundgesetz vorkommt gleich 0.057, aber die Wahrscheinlichkeit für die Reihenfolge von 10 "a"s ist nicht 0.057^{10} sondern 0. In der Praxis muß man für jedes m eine neue Statistik für Σ^m konstruieren.

3 Arithmetische Codes

Im vorherigen Kapitel haben wir gezeigt, daß die erweiterte Huffman-Codierung die Ausgabe mit sehr kleiner Redundanz produziert. Die Idee war: Statt einzelne Symbole des Quellenalphabets codieren wir die Sequenzen der Symbole mit Hilfe des grundlegenden Huffman-Algorithmus. Der Nachteil des Verfahrens ist aber, daß der Algorithmus Huffman-Codes für alle möglichen Sequenzen der Quellensymbole konstruieren muß, um einen Eingabetext zu codieren. Das heißt beispielsweise, daß für ein Quellenalphabet mit 256 Symbolen und für Sequenzen der Länge 3 der Algorithmus 16 777 216 neue Symbole betrachten muß. Deshalb ist der erweiterte Huffman-Algorithmus schon für den Fall der Länge 3 einfach unbrauchbar. In diesem Kapitel werden wir die Arithmetischen Codes diskutieren. Für diese Codes betrachten wir auch statt einzelner Symbole Sequenzen von Symbolen und dann codieren wir diese Sequenzen. Der Vorteil des Verfahrens ist, daß jede Sequenz separat codiert werden kann. Das Schema des Verfahrens ist folgendes: Für gegebenen Text $u_1u_2u_3u_4\dots u_t \in \Sigma^*$, wobei Σ das Quellenalphabet ist, betrachte die Sequenzen der Symbole $u_{mi+1}u_{mi+2}\dots u_{mi+m}$, für $i = 0, 1, \dots, t/m$ (nehmen wir an, daß m t teilt). Dann berechne für jede Sequenz eine numerische Repräsentation – einen Bruch zwischen 0 und 1, und für diese Repräsentation berechne den binären Code. Der Algorithmus, der Arithmetische Codes konstruiert, produziert für jede Sequenz direkt den binären Code, aber um die Methode leichter zu verstehen, werden wir sie weiterhin als zweistufiges Verfahren darstellen:

Sequenz der Symbole $\longrightarrow$ numerische Repräsentation $\longrightarrow$ binärer Code.

3.1 Die numerische Repräsentation

Sei $\Sigma = \{a_1, a_2, a_3, \dots, a_n\}$ das Quellenalphabet mit Wahrscheinlichkeiten $P(a_1), \dots, P(a_n)$. Wir definieren $F(0) = 0$ und für $i = 1, 2, \dots, n$

$$F(i) = \sum_{k=1}^{i} P(a_k).$$

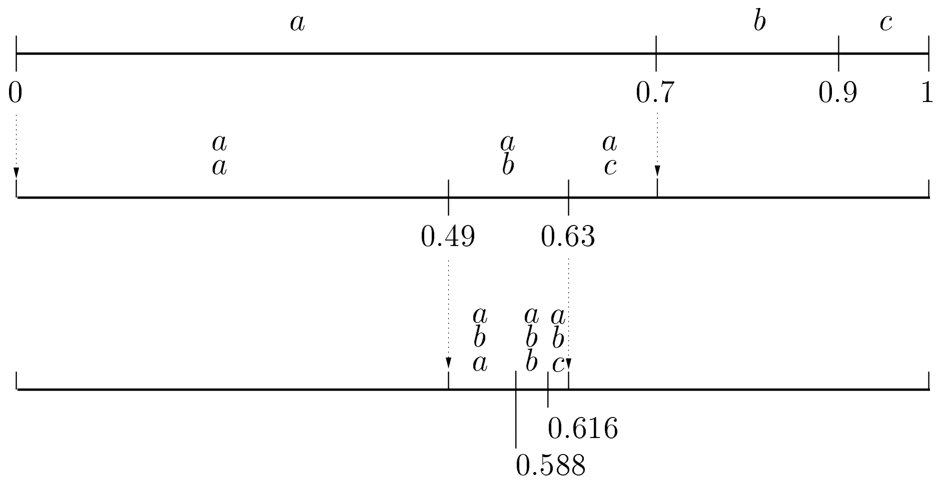
Sei $a_{i_1}a_{i_2}a_{i_3}\dots a_{i_m}$ eine Sequenz. Eine numerische Repräsentation für diese Sequenz ist ein Bruch im Intervall $[l^{(m)}, u^{(m)})$. Die Grenzen des Intervalls definieren wir rekursiv folgendermaßen:

$$l^{(1)} = F(i_1 - 1) \qquad u^{(1)} = F(i_1)$$

und für alle $k = 2, 3, \dots, m$

$$\begin{aligned} l^{(k)} &= l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F(i_k - 1) \\ u^{(k)} &= l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F(i_k). \end{aligned}$$

Als Beispiel für diese Konstruktion betrachten wir $\Sigma = \{a, b, c\}$ mit Wahrscheinlichkeiten $P(a) = 0.7, P(b) = 0.2, P(c) = 0.1$ und $m = 3$. Das Intervall z.B. für die Sequenz *abb* ist $[0.588, 0.616)$ – siehe unten.



Die numerische Repräsentation kann man jetzt beispielsweise folgendermaßen definieren:

$$\overline{T}(a_{i_1} a_{i_2} \ldots a_{i_m}) = \frac{u^{(m)} - l^{(m)}}{2},$$

also der Mittelpunkt des Intervalls. Wir sehen, daß die einzige Information, die der Algorithmus braucht, um die numerische Repräsentation für eine gegebene Sequenz zu berechnen, die Funktion F ist. Insbesondere braucht er nicht die Repräsentationen für die anderen Sequenzen zu kennen.

Um die numerische Repräsentation zu finden, berechnen wir die Grenzen $l^{(k)}$ und $u^{(k)}$, für $k = 1, 2, \ldots, m$ und dann wählen wir einen Wert in $[l^{(m)}, u^{(m)})$. Der umgekehrte Prozeß – eine numerische Repräsentation zu entziffern – läuft sehr ähnlich: Es sei T die gegebene numerische Repräsentation. Der folgende Algorithmus konstruiert die Sequenz für T.

```
Sei l^(0) = 0 und u^(0) = 1;
for k := 1 to m do
begin
   T* := (T - l^(k-1))/(u^(k-1) - l^(k-1));
   Finde i_k so daß F(i_k - 1) <= T* < F(i_k);
   Return(a_{i_k});
   Aktualisiere l^(k) und u^(k);
end.
```

3.2 Binärer Code

Nehmen wir an, daß $\overline{T}(a_{i_1}a_{i_2}\dots a_{i_m}) = (u^{(m)} - l^{(m)})/2$ eine numerische Repräsentation für die Sequenz $a_{i_1}a_{i_2}\dots a_{i_m}$ ist. Die binäre Darstellung der Repräsentation kann natürlich beliebig lang bzw. unendlich sein. In diesem Kapitel zeigen wir, wie man die Repräsentationen mit Hilfe einer kleinen Anzahl von Bits codieren kann.

Es sei, wie früher, $P(a_{i_1}a_{i_2}\dots a_{i_m}) = P(a_{i_1})\cdot P(a_{i_1})\dots P(a_{i_m})$. Wir definieren

$$\ell(a_{i_1}a_{i_2}\dots a_{i_m}) \;=\; \left\lceil \log \frac{1}{P(a_{i_1}a_{i_2}\dots a_{i_m})}\right\rceil + 1.$$

Den binären Code der Repräsentationen definieren wir als $\ell(a_{i_1}a_{i_2}\dots a_{i_m})$ höchstwertige Bits des Bruchs $\overline{T}(a_{i_1}a_{i_2}\dots a_{i_m})$.

Beispiel Sei $\Sigma = \{a,b\}$, $P(a) = 0.9$, $P(b) = 0.1$ und die Länge $n = 2$. Dann bekommen wir folgende Codierung für alle Sequenzen:

x	$\overline{T}(x)$	binär	$\ell(x)$	Code	offenbar besser
aa	0.405	0.0110011110...	2	01	0
ab	0.855	0.1101101011...	5	11011	10
ba	0.945	0.1111000111...	5	11110	110
bb	0.995	0.1111111010...	8	11111110	111

Wir zeigen jetzt, daß man die Codierung eindeutig codieren und decodieren kann. Dann geben wir den Algorithmus für die Codierung bzw. Decodierung an.

Es sei $\lfloor y\rfloor_\ell$ ein Bruch y, der nach dem l-ten Bit abgeschnitten ist. Wir wollen zeigen, daß $\lfloor\overline{T}(x)\rfloor_{\ell(x)}$ ein eindeutiger Repräsentant für die numerische Repräsentation $F(x)$ ist. Äquivalent wollen wir zeigen, daß sich $\lfloor\overline{T}(x)\rfloor_{\ell(x)}$ im Intervall $\left[l^{(m)}, u^{(m)}\right)$ der Sequenz x befindet.

Satz Sei $\left[l^{(m)}, u^{(m)}\right)$ ein Intervall für die Sequenz $x \in \Sigma^m$. Dann

$$\left\lfloor\overline{T}(x)\right\rfloor_{\ell(x)} \in \left[l^{(m)}, u^{(m)}\right).$$

Beweis Wir haben $\left\lfloor\overline{T}(x)\right\rfloor_{\ell(x)} < u^{(m)}$, weil $\overline{T}(x) < u^{(m)}$. Um die zweite Ungleichung zu zeigen, betrachten wir zuerst

$$\begin{aligned}\frac{1}{2^{\ell(x)}} &= \frac{1}{2^{\log(1/P(x))+1}}\\ &\leq \frac{1}{2^{\log(1/P(x))+1}}\\ &= \frac{1}{2^{\frac{1}{P(x)}}} = \frac{P(x)}{2}.\end{aligned} \tag{1}$$

Es gilt also, daß $u^{(m)} - l^{(m)} = P(x)$. Deshalb ist

$$\overline{T}(x) = l^{(n)} + \frac{P(x)}{2}.$$

Dann bekommen wir mit Ungleichung (1)

$$\overline{T}(x) \geq l^{(m)} + \frac{1}{2^{\ell(x)}}.$$

Wenn wir also den Bruch $\overline{T}(x)$ hinter dem $\ell(x)$-ten Bit abschneiden, dann gilt

$$\left\lfloor \overline{T}(x) \right\rfloor_{\ell(x)} \geq l^{(m)}$$

□

Jetzt zeigen wir, daß man den binären Code eindeutig decodieren kann. Diese Eigenschaft folgt direkt aus

Satz Arithmetischer Code ist Präfix-Code.

Beweis Es sei $a \in [0,1)$ und $[b_1, b_2 \ldots b_n]$ die binäre Darstellung für a. Beobachten Sie zuerst, daß die folgende Eigenschaft gilt: Ein Bruch $b \in [0,1)$ hat eine binäre Darstellung mit Präfix $b_1 b_2 \ldots b_n$ genau dann, wenn $b \in [a, a+\frac{1}{2^n})$. Alles, was man jetzt zum Beweis des Satzes noch zeigen muß, ist: Das ganze Intervall $\left[\left\lfloor \overline{T}(x) \right\rfloor_{\ell(x)}, \left\lfloor \overline{T}(x) \right\rfloor_{\ell(x)} + \frac{1}{2^{\ell(x)}}\right)$ liegt in $\left[l^{(m)}, u^{(m)}\right)$, wobei $l^{(m)}$ und $u^{(m)}$ die Grenzen für die Sequenz x sind.
Wir haben schon gezeigt, daß $\left\lfloor \overline{T}(x) \right\rfloor_{\ell(x)} \geq l^{(m)}$. Wir brauchen also nur noch die zweite Ungleichung zu beweisen. Wir haben:

$$\begin{aligned} u^{(m)} - \left\lfloor \overline{T}(x) \right\rfloor_{\ell(x)} &\geq u^{(m)} - \overline{T}(x) \\ &= \frac{P(x)}{2} \geq \frac{1}{2^{\ell(x)}}. \end{aligned}$$

□

Jetzt schätzen wir ab, wie effizient diese Codierung ist. Man braucht, um eine Sequenz x zu codieren, $\ell(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ Bits. Es sei L_{A^m} die erwartete Länge für einen Arithmetischen Code für eine Sequenz der Länge m. Dann gilt:

$$\begin{aligned} L_{A^m} &= \sum P(x)\ell(x) \\ &\leq -\sum P(x) \log P(x) + 2 \sum P(x) \\ &= H(\Sigma^m) + 2. \end{aligned}$$

Weil L_{A^m} immer größer als die Entropie ist, bekommen wir

$$H(\Sigma^m) \leq L_{A^m} \leq H(\Sigma^m) + 2.$$

Für die durchschnittliche Länge L_A pro Symbol bekommen wir

$$\frac{H(\Sigma^m)}{m} \leq L_A \leq \frac{H(\Sigma^m)}{m} + \frac{2}{m}$$

und weil $H(\Sigma^m) = mH(\Sigma)$, gilt

$$H(\Sigma) \leq L_A \leq H(\Sigma) + \frac{2}{m}.$$

Wenn wir also die Länge der Sequenz vergrößern, dann kommen wir immer näher an die Entropie.

4 Wörterbuch-Techniken

Die Idee für Wörterbuch-Techniken ist folgende: Konstruiere eine Liste der Muster, die im Text vorkommen, und codiere die Muster als Indizes der Liste. Diese Methode ist besonders nützlich, wenn sich eine kleine Anzahl Muster sehr häufig im Text wiederholt. Dann gibt es zwei Möglichkeiten: Wenn wir genügend Kenntnisse über den Text haben, dann benutzen wir das statische Verfahren, sonst die dynamische Methode.

4.1 Statisches Verfahren

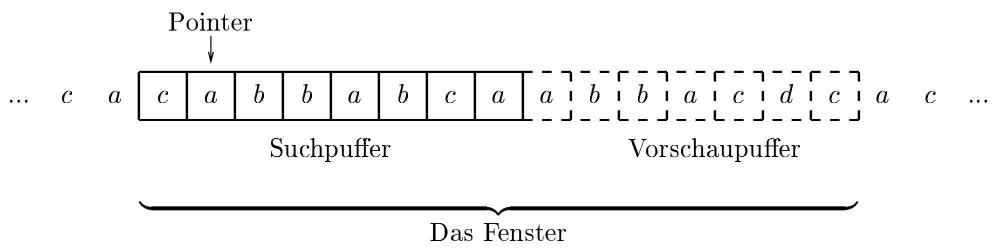
Die meisten statischen Verfahren sind nur für ganz besondere Fälle nützlich. Zum Beispiel für die Kompression der Files, in denen die Leistungen von Studenten gespeichert sind. Dort kommen Wörter wie "Name", "Matrikelnummer" und "Note" sehr oft vor. Hier beschreiben wir aber eine Methode, die etwas allgemeiner angewandt werden kann, nämlich den digram-coding Algorithmus. Für dieses Verfahren enthält das Wörterbuch alle einzelnen Buchstaben und dann möglichst viele Paare von Buchstaben, die wir digrams nennen. Z.B. für das Wörterbuch der Länge 256 sind die ersten 95 Einträge die druckbaren ASCII-Zeichen und die restlichen 161 Einträge sind die Paare der Symbole, die am häufigsten benutzt worden sind. Dann codiert man jedes Symbol bzw. jedes Symbol-Paar mit 8 Bits.

4.2 Dynamisches Wörterbuch

Alle wichtigen Verfahren dieser Familie basieren auf zwei Algorithmen, die von Jacob Ziv und Abraham Lempel 1977 bzw. 1978 beschrieben wurden, nämlich LZ77 und LZ78. Hier beschreiben wir die grundlegenden Algorithmen und ihre Varianten:

- LZSS – Lempel-Ziv-Storer-Szymanski (gzip, ZIP und andere),
- LZW – Lempel-Ziv-Welch,
- LZC – Lempel-Ziv Compress und
- LZMW (Unix Compress, GIF).

LZ77 (auch *sliding windows* Algorithmus genannt) bearbeitet den Text durch ein Fenster, das durch den Text verschoben wird. Das Fenster hat zwei Teile: den Suchpuffer und den Vorschaupuffer. Der erste Puffer enthält das Fragment des Textes, der bis zu einem bestimmten Zeitpunkt verarbeitet wurde und die Zeichenfolge im Puffer dient als Wörterbuch, um die Zeichen im Vorschaupuffer zu codieren. Die Länge des Puffers von vornherein festgelegt. Als Beispiel siehe Abbildung unten, in der die Länge des Puffers 8 bzw. 7 ist.



Um einen neuen Code zu konstruieren, benutzt LZ77 das folgende Verfahren:

1. Verschiebe den Pointer auf das erste Symbol von rechts im Suchuffer. Dann sei `Startsymbol, S :=` ”das erste Symbol (von links) im Vorschauuffer” und `offset, L := 0`.

2. Verschiebe den Pointer im Suchpuffer so weit nach links, bis er das Zeichen findet, das gleich dem `Startsymbol` ist, oder die linke Grenze des Puffers erreicht hat. Wenn die Grenze des Puffers erreicht ist, dann gib als Code

   `(offset, L, K(S))`

   aus, wobei `K(S)` der Code des Zeichens `S` ist; sonst gehe zum nächsten Schritt.

3. Lies Zeichen für Zeichen gleichzeitig vom `Startsymbol` und Pointer so lange bis die Zeichen gleich sind. Wenn die Länge der durchgelesenen Zeichenfolgen größer als `L` ist, dann nimm diese Länge als neuen Wert für `L` und den Abstand des Pointers zur rechten Grenze des Suchpuffers als neuen Wert für `offset`. Weiterhin nimm das `L+1`-ste Symbol des Vorschaupuffers als `S` und beginne so wieder mit Schritt 2.

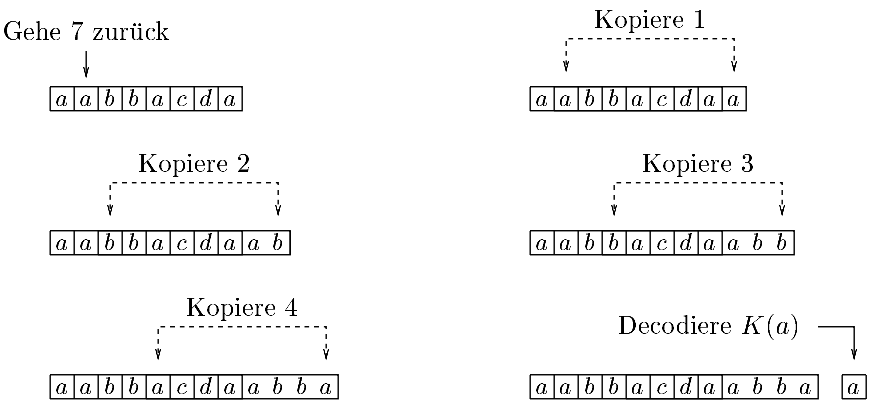
Wenn der Code fertig ist, dann verschiebt man das ganze Fenster um `L+1` Symbole, um einen neuen Code zu konstruieren. Als Beispiel betrachten wir, wie der Algorithmus den Text: *abaabbabaacaacabb* codiert. Das Zeichen `eof` steht für das Ende des Textes.

Inhalt der Puffer	*Generierter Code*
$\boxed{}\boxed{a\ b\ a\ a\ b}\ b\ a\ b\ a\ a\ c\ a\ a\ c\ a\ b\ b$	$(0, 0, K(a))$
$\boxed{a}\boxed{b\ a\ a\ b\ b}\ a\ b\ a\ a\ c\ a\ a\ c\ a\ b\ b$	$(0, 0, K(b))$
$\boxed{a\ b}\boxed{a\ a\ b\ b\ a}\ b\ a\ a\ c\ a\ a\ c\ a\ b\ b$	$(2, 1, K(a))$
$\boxed{a\ b\ a\ a}\boxed{b\ b\ a\ b\ a}\ a\ c\ a\ a\ c\ a\ b\ b$	$(3, 1, K(b))$
$\boxed{a\ b\ a\ a\ b\ b}\boxed{a\ b\ a\ a\ c}\ a\ a\ c\ a\ b\ b$	$(6, 4, K(c))$
$a\ b\ a\ a\ b\ \boxed{b\ a\ b\ a\ a\ c}\boxed{a\ a\ c\ a\ b}\ b$	$(3, 4, K(b))$
$a\ b\ a\ a\ b\ b\ a\ b\ a\ a\ \boxed{c\ a\ a\ c\ a\ b}\boxed{b}$	$(1, 1, K(\texttt{eof}))$

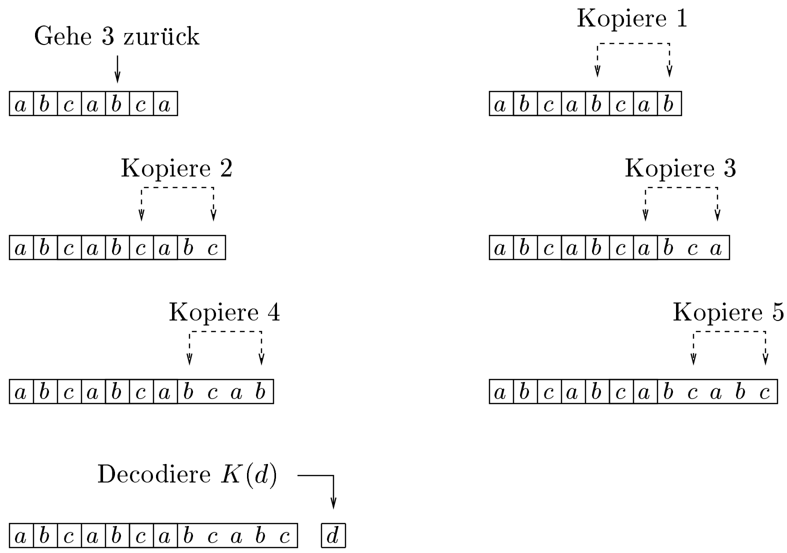
Beobachten Sie eine interessante Sache, die in Zeile 6 vorkommt: Die Zeichenfolge der Länge 4, die mit `offset` $= 3$ im Suchpuffer anfängt, beendet sich im Vorschaupuffer. Natürlich ist eine solche Situation sehr günstig, weil je länger die Zeichenfolgen desto kürzer der gesamte Code. Das Problem ist aber: Können wir solche 'überlappende' Zeichenfolgen eindeutig decodieren? Wir zeigen unten, daß das Decodierungsverfahren für LZ77 auch in solchen Fällen korrekt funktioniert.

Das Decodierungsverfahren läuft ganz natürlich. Anstelle einer formalen Beschreibung zeigen wir zwei Beispiele, in denen die Länge der Suchpuffer 7 ist. Nehmen wir an, daß der Inhalt des Suchpuffers sowie der nächste Code die Eingaben des Verfahrens sind.

Beispiel Der Inhalt des Suchpuffers ist: *aabbacda* und der Code: $(7, 4, K(a))$.



Beispiel Der Inhalt des Absuchen-Puffer ist: *abcabca* und der Code: $(3, 5, K(d))$ (die Decodierung für eine 'überlappende' Zeichenfolge).



LZSS ist eine Variante von LZ77, die im Ausgabecode keinen Code für das nächste Zeichen angibt. Jetzt hat jeder Ausgabecode eins von zwei Formaten:

$$(0, \text{offset, Länge}),$$

$$(1, \text{Zeichen}).$$

Typischerweise konstruiert LZSS den Code des ersten Formats, wenn die Länge größer als 2 ist und dann benutzt der Algorithmus Huffman-Codierung, um die Reihenfolge von 'Ausgabecode-Symbolen' zu codieren. In der Praxis ist die Länge des Fensters gleich 32.

Beispiele:
Eingabe: $a\ a\ a\ a\ a\ a$
Ausgabecode: $(1, a)$ $(0, 1, 6)$ $(1, \texttt{eof})$

Eingabe: $a\ a\ a\ b\ a\ b\ a\ b\ a$
Ausgabecode: $(1, a)$ $(1, a)$ $(1, a)$ $(1, b)$ $(0, 2, 5)$ $(1, \texttt{eof})$

LZ77 (und seine Variante) bearbeitet besonders schlecht Eingaben, in denen sich Zeichenfolgen befinden, die sich im Eingabetext in größerem Abstand wiederholen als die Länge des Fensters erfassen kann. Zum Beispiel im Text:

$$\ldots\ e\ f\ g\ h\ i\ a\ b\ c\ d\ e\ f\ g\ h\ i\ a\ \boxed{b\ c\ d\ e\ f\ g\ h\ i}\boxed{a\ b\ c\ d\ e\ f\ g\ h}\ i\ \ldots$$

Der Algorithmus kann die Zeichenfolgen, die sich im Text oft wiederholen, nicht komprimieren (wegen der Länge des Fensters) und er konstruiert für diesen Fall eine sehr uneffiziente Ausgabe. Die nächste Algorithmen werden die Texte mit solchen Eigenschaften viel besser bearbeiten.

Wir betrachten zuerst das grundlegende **LZ78** Komprimierungsverfahren. Dieser Algorithmus speichert die Informationen über die Zeichenfolgen, die er im Eingabetext findet, in der Matrize `Wörterbuch`. Am Anfang ist diese Matrize leer und dann speichert der Algorithmus im `Wörterbuch` sukzessiv neue Zeichenfolgen und gleichzeitig generiert er die Ausgabecodes für diese Folgen. Jeder Code hat das Format: (i, k), wobei i der Index zum `Wörterbuch` ist und k der Code eines einzelnen Zeichens. Das heißt, daß jedes Paar (i, k) eine Zeichenfolge codiert, die eine Verkettung zweier Wörter ist: das erste Wort mit der Länge ≥ 0 und das zweite mit der Länge 1. Um den neuen Code zu generieren, findet der LZ78-Algorithmus im Eingabetext die längste Zeichenfolge w, die schon im `Wörterbuch` gespeichert wurde, und nimmt als i den Index für w. Wenn das Wort w leer ist, dann nehmen wir $i = 0$ an. Dann liest der Algorithmus das nächste Zeichen Z, fügt wZ dem `Wörterbuch` hinzu und gibt $(i, K(Z))$ aus. Für $K(Z)$ nehmen wir den Code für das Zeichen Z an solange bis der Algorithmus das erste Mal die Zeichenfolge wZ betrachtet, wobei w ein Leerwort ist. Dann ist $K(Z)$ gleich dem Index der Matrize `Wörterbuch` für dieses Zeichen. Das folgende Beispiel zeigt wie der LZ78-Algorithmus mit der Eingabe *ababab cabac* funktioniert.

	`Wörterbuch`		Ausgang
Text	Index	Eintrag	Code
a	1	a	$(0, K(a))$
b	2	b	$(0, K(b))$
a			
b	3	ab	$(1, 2)$
a			
b			
c	4	abc	$(3, K(c))$
a			
b			
a	5	aba	$(3, 1)$
c	6	c	$(0, K(c))$
`eof`			

Der Dekomprimierungsalgorithmus für die LZ78-Methode läuft ganz einfach: Er startet mit der leeren Matrize `Wörterbuch` und dann rekonstruiert er sie.

Das **LZW** Komprimierungsverfahren, das Terry Welch 1984 entwickelt hat, ist eine Variante von LZ78. LZW gibt im Ausgabecode keinen Code für die Zeichen an und als Codes benutzt der Algorithmus direkt die Indizes des Wörterbuches. Das Verfahren startet mit dem Wörterbuch, das die einzelnen Standardzeichen enthält. Nehmen wir an, daß wir 256 solche Zeichen betrachten werden und daß die Indizes der Matrize `Wörterbuch` mit 0 anfangen. Der Encode-Algorithmus sieht folgendermaßen aus:

```
w = NIL
while (Z = readchar()) {
  if wZ in Wörterbuch
    w = wZ
  else {
    gib Code für w aus
    füge wZ ins Wörterbuch ein
    w = Z }
}
gib Code für w aus
```

Eine wichtige Eigenschaft ist, daß der Encode-Algorithmus einen Code für ein Wort `wZ` genau dann ausgibt, wenn er früher mindestens einmal das Wort im Text getroffen hat. Diese Eigenschaft spielt eine entscheidende Rolle bei der Decodierung.

Das folgende Beispiel zeigt, wie der Algorithmus mit der Eingabe *abababcabac* funktioniert (vergleichen Sie LZW mit LZ78 bei gleicher Eingabe, das haben wir früher betrachten).

Text	`w`	`Z`	Neuer Eintrag im `Wörterbuch` Eintrag	Index	Ausgang Code
a		*a*			
b	*a*	*b*	*ab*	256	*a*
a	*b*	*a*	*ba*	257	*b*
b	*a*	*b*			
a	*ab*	*a*	*aba*	258	(256)
b	*a*	*b*			
c	*ab*	*c*	*abc*	259	(256)
a	*c*	*a*	*ca*	260	*c*
b	*a*	*b*			
a	*ab*	*a*			
c	*aba*	*c*	*abac*	261	(258)
`eof`	*c*				*c*

Das zweite Beispiel zeigt wie der LZW einen Text mit vielen überlappenden Zeichenfolgen codiert.

Text	`w`	`Z`	Neuer Eintrag im `Wörterbuch` Eintrag	Index	Ausgang Code
a		*a*			
a	*a*	*a*	*aa*	256	*a*
a	*a*	*a*			
a	*aa*	*a*	*aaa*	257	(256)
a	*a*	*a*			
a	*aa*	*a*			
a	*aaa*	*a*	*aaaa*	258	(257)
`eof`	*a*				*a*

Um den Code zu dekomprimieren, startet der Algorithmus mit der Matrize `Wörterbuch`, die die einzelnen Standardzeichen enthält (wie beim Encode-Verfahren) und dann rekonstruiert er die Matrize. Der Decode-Algorithmus sieht folgendermaßen aus:

```
v = Wörterbuch(readcode())
output v
```

```
while (C = readcode()) {
  if C in Wörterbuch
    w = Wörterbuch(C)
  else {
    K = v[0]  v[0] ist das erste Zeichen in v
    w = vK
  }
  output w
  Z = w[0]
  füge vK ins Wörterbuch ein
  v = w
}
```

Über die Benutzung von Tries in LZW (Beispiele).

5 Verlustfreie Komprimierung von Bildern

Die große Effizienz der Arithmetischen Codierung für Texte folgt daraus, daß die Verteilung der Zeichenfolgen (konstanter Länge) ungleichmäßig verteilt ist. Auch die Wörterbuch-Techniken nutzen die Eigenschaft von Texten, daß sich viele Muster sehr häufig wiederholen. Um diese Methoden zu entwickeln, brauchten wir deshalb nicht nach neuen Modellen für Texte zu suchen. Leider haben andere Datentypen, wie z.B. Stimme oder Bilder, keine so schönen Eigenschaften wie Texte. Bilder beispielsweise (oder genauer gesagt die Files mit den Bitmaps der Bilder) haben eine ganz andere Struktur als Textfiles. Ein Grund dafür ist einfach: Die Sprache (und insbesonders eine künstliche Sprache, wie z.B. eine Programmiersprache) wird mit Regeln generiert, die einen bestimmten Einfluß auf die Struktur des Textfiles haben. Im Fall der Bilder haben wir es mit ganz anderen Regeln zu tun. Deshalb brauchen wir, um die Komprimierungsverfahren für Bildern zu entwickeln, zuerst ein Modell, das den Eigenschaften der Bilder besser entspricht als ein Quellenalphabet mit Verteilung für Zeichen (Pixel).

In diesem Kapitel beschreiben wir das Markov Modell für binäre Bilder und dann betrachten wir (lossless) Facsimile Encoding und die fortschreitende Bildübertragung (*progressive image transmission*).

5.1 Die bedingte Entropie und das Markov Modell

In dem Kapitel diskutieren wir die bedingte Entropie – einen Begriff, den wir für verlustfreie Komprimierung von Bildern benutzen werden. Später zeigen wir, daß der Begriff auch für verlustbehaftete Komprimierung sehr nützlich ist.

Seien A und B zwei Ereignisse und $P(A|B)$ die bedingte Wahrscheinlichkeit, daß A eintritt unter der Bedingung, daß auch B eintritt. Dann definieren wir den bedingten Informationsgehalt folgendermaßen:

$$i(A|B) = \log_2 \frac{1}{P(A|B)}.$$

Beobachten Sie, daß, wie im Fall des Informationsgehaltes, der Begriff intuitiv klar ist. Sei jetzt S eine bestimmte Informationsquelle und T ein Kontext, in dem die Nachrichten aus S vorkommen. Dann ist die bedingte Entropie $H(S|T)$ der "mittlere Informationsgehalt pro Nachricht aus S mit dem Kontext T":

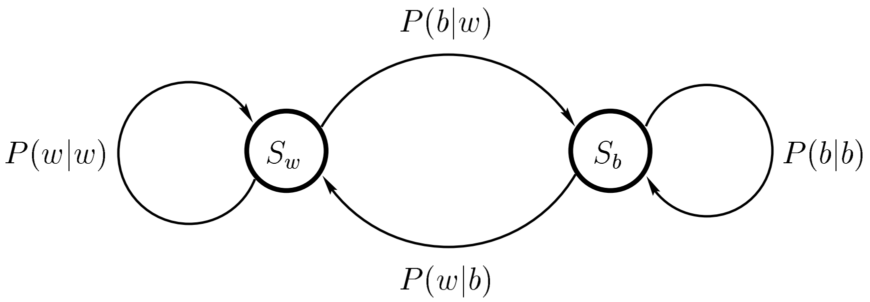
$$H(S|T) = \sum_{t \in T} P(t) \sum_{s \in S} P(s|t) \log_2 \frac{1}{P(s|t)}.$$

Wie früher, werden wir meistens S als endliches Quellenalphabet Σ betrachten. Als Kontext T werden wir eine endliche Teilmenge von Σ^* annehmen. Es ist nicht schwer zu zeigen, daß, wenn S unabhängig vom Kontext ist, dann $H(S|T) = H(S)$ gilt; ansonsten haben wir: $H(S|T) < H(S)$. Die Kenntnis des Kontextes für S reduziert also die Entropie.

Wir stellen diese bedingten Wahrscheinlichkeiten mit Hilfe des Markov-Modells dar. Um das Konzept zu verstehen, betrachten wir das folgende Beispiel. Nehmen wir an, daß das Bild nur zwei Sorten von Pixeln hat: weiß und schwarz. Das heißt, unser Quellenalphabet ist $\{w, b\}$. Als Kontext für ein Pixel x werden wir einfach das Pixel betrachten, das eine Position früher in der Zeile steht. Entsprechende Wahrscheinlichkeiten sind folgende:

$$\begin{aligned}
P(w|w) &= \text{Wahrscheinlichkeit ein weißes Pixel zu erhalten,}\\
&\ \text{wenn das vorherige Pixel weiß war;}\\
P(w|b) &= \text{Wahrscheinlichkeit ein weißes Pixel zu erhalten,}\\
&\ \text{wenn das vorherige Pixel schwarz war;}\\
P(b|b) &= \text{Wahrscheinlichkeit ein schwarzes Pixel zu erhalten,}\\
&\ \text{wenn das vorherige Pixel schwarz war;}\\
P(b|w) &= \text{Wahrscheinlichkeit ein schwarzes Pixel zu erhalten,}\\
&\ \text{wenn das vorherige Pixel weiß war.}
\end{aligned}$$

Diese Wahrscheinlichkeiten können wir als Markov-Modell folgendermaßen darstellen. Hier entspricht S_w, bzw. S_b, dem Zustand, daß wir ein weißes, bzw. schwarzes Pixel lesen:



Betrachten wir jetzt ein ganz konkretes Beispiel: Nehmen wir die folgenden Wahrscheinlichkeiten an

$$\begin{array}{ll}
P(w) = P(S_w) = 0.85 & P(b) = P(S_b) = 0.15\\
P(w|w) = 0.99 & P(b|w) = 0.01\\
P(w|b) = 0.2 & P(b|b) = 0.8.
\end{array}$$

Dann ist die Entropie für das Quellenalphabet $\Sigma = \{w, b\}$ gleich

$$H(\Sigma) \;=\; -0.85 \times \log 0.85 \;-\; 0.15 \times \log 0.15 \;=\; 0.610.$$

Also ist für dieses Modell (wobei wir keine Korrelation zwischen Pixeln beachten) ist die (bestmögliche) mittlere Anzahl von Bits gleich 0.61, um die Nachricht aus Σ zu kodieren. Wir zeigen, daß diese Zahl für das Markov-Modell viel besser ist. Die entsprechende bedingte Entropie berechnen wir folgendermaßen:

$$\begin{aligned} H(\Sigma|\Sigma) &= 0.85 \times (-0.99 \times \log 0.99 \;-\; 0.01 \times \log 0.01) \;+ \\ &\; 0.15 \times (-0.2 \times \log 0.2 \;-\; 0.8 \times \log 0.8) \\ &= 0.85 \times 0.081 + 0.15 \times 0.722 \;=\; 0.176. \end{aligned}$$

Für das Markov Modell ist die mittlere Anzahl der Bits gleich 0.176, um etwa 71% besser als früher. Die Codierung sieht folgendermaßen aus: Wir konstruieren separat eine Codierung für Σ im Kontext des weißen Pixels und eine Codierung für Σ im Kontext des schwarzen Pixels. Dann benutzen wir, um die Pixel zu codieren, so lange wir im Zustand S_w bleiben, die erste Codierung; sonst die zweite.

5.2 Facsimile Codierung (Faxen)

Die Aufgabe ist: Taste eine A4 Seite ab und komprimiere das Schwarzweiß-Bild, um es durch die Telefonleitung zu senden. Eine interessante Beobachtung ist, daß die ersten Anwendungen von Komprimierungsverfahren überhaupt gerade für Facsimile benutzt worden sind. Diese Verfahren benutzen das Markov-Modell mit zwei Zuständen: S_w und S_b, wie wir es im vorherigen Kapitel betrachtet haben. In dem Kapitel beschreiben wir zwei Methoden, die in der Praxis benutzt werden: das 1-dimensionale Schema **MH** (*modified Huffman*) und das 2-dimensionale Schema **MR** (*modified READ*, wobei READ *relative element address designate* bedeutet).

Der MH Algorithmus ist ganz einfach: Wir codieren mit Hilfe der Huffman-Codierung die *Lauflängen* (engl. *run*), wobei ein Lauf die längste Reihenfolge des gleichen Pixels ist. Wir geben diese Länge für jede Zeile separat an und vermuten dann, daß die erste Zahl den Lauf der weißen Pixel kodiert. Weil die Anzahl der Pixel pro Zeile sehr groß ist (1728), können die Läufe auch so eine große Anzahl von verschiedenen Längen haben. Um also die Huffman Methode effizient zu nutzen, codieren wir jede Länge des Laufes r_l, der länger als 64 ist, folgendermaßen:

$$r_l = 64 \times m + t \qquad \text{for} \;\; m = 1, 2, \ldots, 27 \;\; \text{und} \;\; t = 0, 1, \ldots, 63$$

und stellen r_l als Paar $64 \times m$ und t dar. Wir nennen Codes für $64 \times m$ – *make-up codes* und Codes für t – *terminating codes* (Schlußcode). Falls $r_l < 63$ benutzen wir nur den Schlußcode, um den Lauf zu codieren. Dann konstruieren wir den Huffman-Code für das Alphabet

$$\{64 \times 1,\ 64 \times 2,\ \ldots,\ 64 \times 27\} \cup \{0, 1, \ldots, 63\}$$

für die weißen Läufe und separat für das gleiche Alphabet

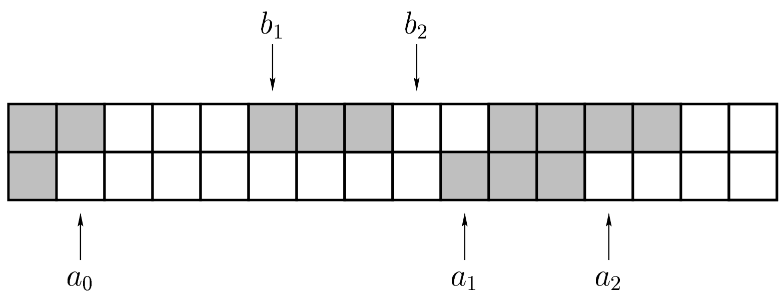
$$\{64 \times 1,\ 64 \times 2,\ \ldots,\ 64 \times 27\} \cup \{0, 1, \ldots, 63\}$$

für die schwarzen Läufe. In dem CCITT-Standard (Consultative Committee on International Telephone and Telegraph, jetzt ITU-T: International Telecommunications Union) hat beispielsweise der schwarze Schlußcode für 2 die Länge 2 und für 63 die Länge 12. Für die weißen Schlußcodes sind diese Längen: 4 bzw. 8 (es ist sehr wahrscheinlich, daß der schwarze Lauf die Länge 2 hat!)

Der MR Algorithmus nutzt noch mehr Korrelationen zwischen Pixeln als die MH Methode. Das MR Verfahren betrachtet nämlich die Abhängigkeiten in zwei Zeilen. Wir nennen das erste Pixel eines Laufes das Übergangspixel. Jetzt nehmen wir an, daß wir eine ganze Zeile codiert (bzw. decodiert) haben und daß wir nun die nächste Pixel-Zeile bearbeiten. Wir definieren folgende Positionen für die Übergangspixel (als Beispiel, siehe Bild unten):

- a_0: Das ist ein Pixel, das wir gerade betrachten. Wir nehmen an, daß jede Zeile mit einem weißen Lauf anfängt (wenn nicht, betrachten wir den Lauf der Länge 0).
- a_1: Das erste Übergangspixel nach rechts von Pixel a_0.
- a_2: Das zweite Übergangspixel nach rechts von Pixel a_0.
- b_1: Das erste Übergangspixel in der vorherigen Zeile, das eine umgekehrte Farbe als Pixel a_0 hat.
- b_2: Das erste Übergangspixel nach rechts von Pixel b_1.

Das Codierungsverfahren kennt alle Positionen. Wenn wir decodieren, kennen wir nur b_1, b_2 und a_0, um das Ziel a_1 und a_2 zu finden.



Der Codier-/Decodier-Algorithmus betrachtet drei Fälle:

1. *Zwischen-Modus*: b_1 und b_2 liegen zwischen a_0 und a_1. Der Code ist 0001. Dann nimm Position a_1 als neuen Wert für a_0, suche neue Werte b_1 und b_2 und geh weiter mit dem ganzen Prozeß.

2. *Vertikal-Modus*: a_1 liegt vor b_2.

   2.1 Abstand zwischen a_1 und b_1 ≤ 3. Dann sind es folgende Codes:

Pos. a_1 bezüglich b_1	Code	Länge des Codes
b_1		
a_1	1	1
$b_1 a_1$	011	3
$b_1 \sqcup a_1$	000011	6
$b_1 \sqcup \sqcup a_1$	0000011	7
$a_1 b_1$	010	3
$a_1 \sqcup b_1$	000010	6
$a_1 \sqcup \sqcup b_1$	0000010	7

   2.2 Abstand zwischen a_1 und b_1 > 3. Dann sieht der Code folgendermaßen aus: erstens 001 und dann MH-Codes für zwei Läufe von a_0 nach $a_1 - 1$ und von a_1 nach $a_2 - 1$.

Das Problem der Fehlerfortpflanzung durch Codierungsabhängigkeiten zwischen den Zeilen wird durch häufiges, regelmäßiges Zwischenschreiben "neuer" MH-Codierungen angegangen. SBIG, eine weitere Empfehlung von CCITT, wird später behandelt.

5.3 Fortschreitende Bildübertragung

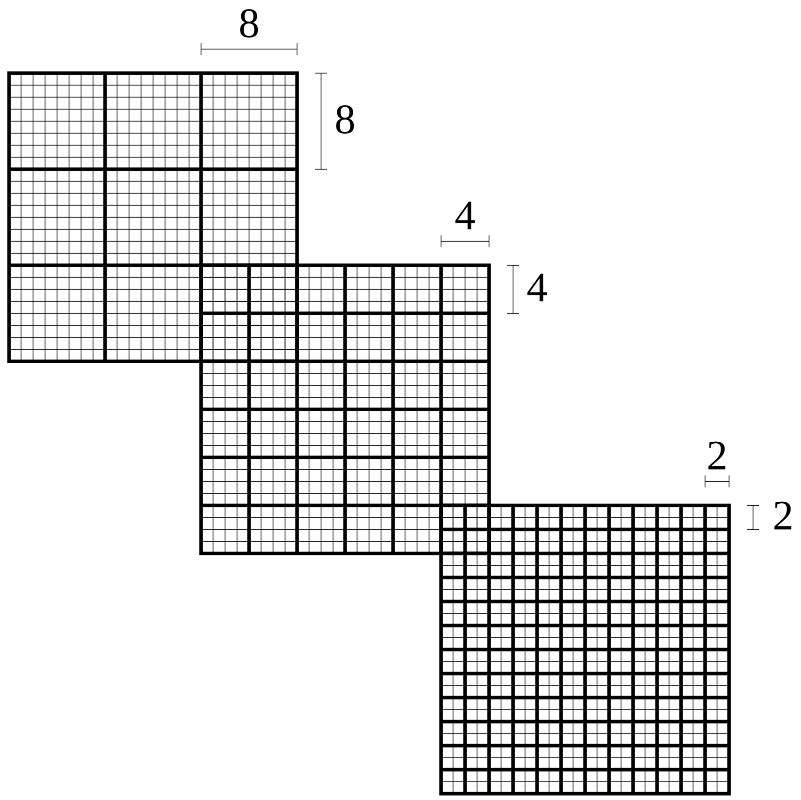
Betrachten wir die folgende Situation: Mit Hilfe des Browsers wollen wir 30 Bilder durchsehen. Wie es sehr häufig in der Praxis passiert, werden nicht alle Bilder für uns gleich wichtig sein: Nur manche werden für uns große Bedeutung haben und nur solche wollen wir auf dem Schirm des Computers haben. Nehmen wir an, daß jedes Bild schwarz-weiß ist und daß es die Größe 512×512 hat mit 8 Bits pro Pixel. Wir wollen diese Bilder durch die Leitung mit der Geschwindigkeit 14.4-kbit/s übertragen und das heißt, daß die ganze Übertragung

$$30 \times (8 \times 512 \times 512)\ /\ 14\,400 \ = \ 62\,914\,560\ /\ 14\,400 \ \approx \ 4\,369 \text{ s} \ \approx \ 73 \text{ min.}$$

dauert.
Jetzt modifizieren wir unser Verfahren so, daß wir statt der Bilder die Approximationen der Bilder übertragen. Die Approximationen haben eine viel

kleinere Größe als die Bilder, aber trotzdem ist es immer möglich, ganz gut zu entscheiden, was das Bild zeigt. Deshalb können wir relativ schnell eine Übertragung für diese Bilder anhalten, die für uns uninteressant sind. Eine solche Übertragung (zuerst die Approximationen und am Ende die lossless Repräsentation der Bilder) heißt fortschreitende Bildübertragung.

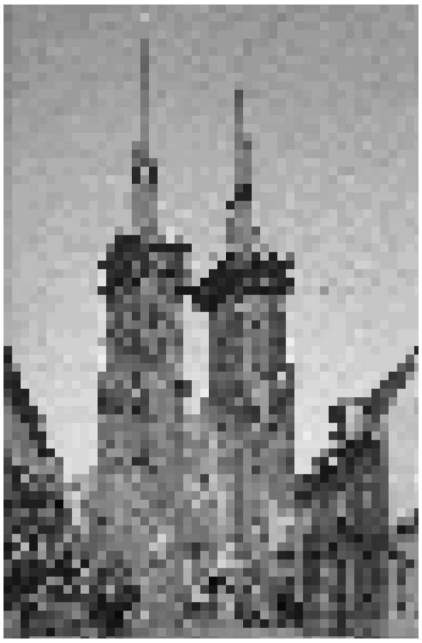
Ein einfaches und schnelles Verfahren, um eine Approximation zu bekommen ist folgendes: wir betrachten $b \times b$ Pixels des Bildes und dann stellen wir alle b^2 Pixel als ein Pixel dar. Das heißt, wir komprimieren mit dem Faktor b^2. Und jetzt, um das Bild zu übertragen, senden wir z.B. drei Approximationen (mit den Parametern $b = 8$, 4 und 2 wie unten) und die lossless Codierung des Bildes nacheinander.

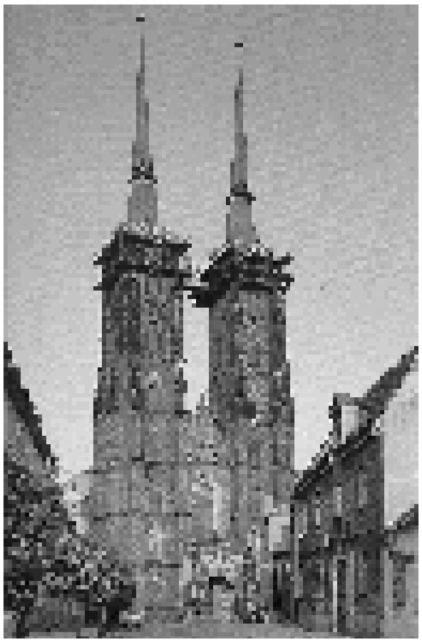


Unten können Sie vergleichen, welche Approximationen wir bekommen, wenn wir (für $b = 4$)als Repräsentant für die $b \times b$ Pixel das im Quadrat links oben gelegene (siehe linkes Bild) oder den Medianwert aller b^2 Pixel (siehe rechtes Bild) nehmen.

Nun das Original:

Wie effizient die fortschreitende Bildübertragung ist, können Sie unten sehen, wo wir dieses Verfahren mit einer traditionellen Methode (verlustfreie Übertragung Zeile für Zeile) vergleichen. Die linken Approximationen haben die gleiche Größe wie die entsprechenden Teile des Originals auf der rechten Seite (die erste Reihe ist für $b = 8$ die nächste für $b = 4$ und unten sehen Sie das Original-Bild).





6 Grundlagen der verlustbehafteten Komprimierung

Ein Verfahren A heißt verlustbehaftete Komprimierung, wenn man für die Eingabe $\mathcal{X}$ nicht von der Repräsentation $\mathcal{X}_c$, die A konstruiert, eine Ausgabe $\mathcal{Y}$ so rekonstruieren kann, daß $\mathcal{X} = \mathcal{Y}$. Solche Methoden benutzen wir, wenn es unmöglich ist, für $\mathcal{X}$ irgendeine 'genaue' Repräsentation $\mathcal{X}_c$ zu bekommen: z.B. wenn $\mathcal{X}$ analog ist (Stimme) und wir die Repräsentation in digitaler Form haben wollen. Solche Methoden sind auch sehr nützlich, wenn wir einen (in der Praxis ganz kleinen) Unterschied zwischen $\mathcal{X}$ und $\mathcal{Y}$ akzeptieren und den Kompressionsquotienten wesentlich kleiner haben wollen, als es die beste verlustfreie Methode bietet. Betrachten wir als Beispiel die folgenden zwei 256×256 Schwarz-Weiß-Bilder, eine Brücke und eine Kamera zeigend. Die linken Bilder sind die Originale, und die rechten Bilder zeigen die Rekonstruktionen, die wir mit Hilfe von verlustbehafteten JPEG-Verfahren bekommen.

JPEG-Verfahren werden wir im Kapitel 9 genau diskutieren. Der Vorteil des Verfahrens sehen Sie unten, wo wir die Größe für die Bit-Map und GIF Format (also für die genaue Repräsentationen von Bildern) mit der Größe der JPEG-Repräsentation vergleichen.

Bild	Bit Map	GIF	JPEG
Brücke	65 536	76 511	17 272
Kamera	65 536	55 484	10 889

In diesem Kapitel diskutieren wir die grundlegenden Begriffe für die verlustbehaftete Komprimierung. Es ist klar, daß wir jetzt nicht nur Kompressionsquotienten für die Algorithmen betrachten sollten, sondern auch ein Maß für den Unterschied zwischen $\mathcal{X}$ und $\mathcal{Y}$. Diesen Unterschied werden wir Verzerrung (engl. *distortion*) nennen.

Sei $\mathcal{X} = \{x_n\}_{n=1}^N$ und $\mathcal{Y} = \{y_n\}_{n=1}^N$. Typische Maßen für die Verzerrung sind:

$$d(x_n, y_n) = (x_n - y_n)^2$$

das sogenannte *quadratische Fehlermaß* ("Methode der kleinsten Quadrate") und

$$d(x_n, y_n) = |x_n - y_n|$$

das *Betragsfehlermaß*. Für diese Maße können wir den Unterschied zwischen $\mathcal{X}$ und $\mathcal{Y}$ folgendermaßen messen:

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - y_n)^2$$

oder auch:

$$d_1 = \frac{1}{N}\sum_{n=1}^{N}|x_n - y_n| \ .$$

σ^2 heißt *mittlerer quadratischer Fehler* und d_1 *mittlerer Betragsfehler*. Für $\mathcal{X}$ und $\mathcal{Y}$, mit $\mathcal{X} = \mathcal{Y}$, haben wir: $\sigma^2 = d_1 = 0$. Für weitere Beispiele sehen Sie die Tabelle unten.

$\mathcal{X}$	$\mathcal{Y}$	σ^2	d_1
Brücke	JPEG-Repräsent. für Brücke	44.20	5.07
Kamera	JPEG-Repräsent. für Kamera	22.34	3.07
Brücke	Kamera	6 674.66	66.93

Als relatives Fehlermaß (Verzerrungsmaß) für verlustbehaftete Komprimierung $\mathcal{Y}$ von $\mathcal{X}$ benutzen wir das Verhältnis von Signal zu Verzerrung (engl. *signal-to-noise ratio*, kurz *SNR*), $SNR = \frac{\sigma_X^2}{\sigma^2}$, wobei σ_X^2 der mittlere quadratische Wert der Quellenausgabe (also des Signals) und σ^2 der mittlere quadratische Fehler ist. SNR wird häufig logarithmisch skaliert in Dezibel angegeben, also

$$SNR(dB) = 10\log_{10}\frac{\sigma_X^2}{\sigma^2}.$$

Es sei jetzt X eine Zufallsvariable für das Eingabesymbol aus dem Quellenalphabet $\{x_0, x_1, \ldots, x_{N-1}\}$ und Y eine Zufallsvariable für das Ausgabesymbol aus $\{y_0, y_1, \ldots, y_{M-1}\}$. Wir nennen $\log[P(y_j|x_i)/P(y_j)]$ *mutual information* und wir definieren *average mutual information* folgendermaßen:

$$I(Y;X) \quad = \quad \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} P(x_i, y_j) \times \log[P(y_j|x_i)/P(y_j)].$$

Es ist einfach zu beweisen, daß die folgende schöne Eigenschaft gilt:

$$I(Y;X) = H(Y) - H(Y|X)\ ,$$

wobei $H(Y|X)$ die bedingte Entropie ist, die wir im Kapitel 5 definiert haben.[1] Dann definieren wir:

$$D = \sum_{i=0}^{N-1} P(x_i) \sum_{j=0}^{M-1} P(y_j|x_i) \times d(x_i, y_j).$$

Wir nennen D die Verzerrung. Die Frage ist jetzt: Wie ist die trade-off zwischen der Verzerrung und dem Kompressionsquotienten oder anders: Wie ist der bestmögliche Kompressionsquotient R für die Verzerrung, die kleiner oder gleich D ist? Wir wollen diese Abhängigkeit durch eine Funktion $R(D)$ beschreiben.

Es sei $\Gamma(D)$ die Menge von Verteilungen für Y so daß:

$$\sum P(x_i) \sum P(y_j|x_i) \times d(x_i, y_j) \ \leq \ D.$$

Das heißt für eine bestimmte Eingabe $P(x_0), P(x_1), \ldots, P(x_{N-1}), D$ ist jede Ausgabe des Kompressionsverfahrens, das die Verteilung für Y aus $\Gamma(D)$ hat, mit der Verzerrung höchstens D. 1959 hat Shannon die folgende wichtige Eigenschaft bewiesen:

$$R(D) = \min_{Y\in\Gamma(D)} I(Y;X).$$

[1] Weitere Informationen finden Sie z.B. in [13].

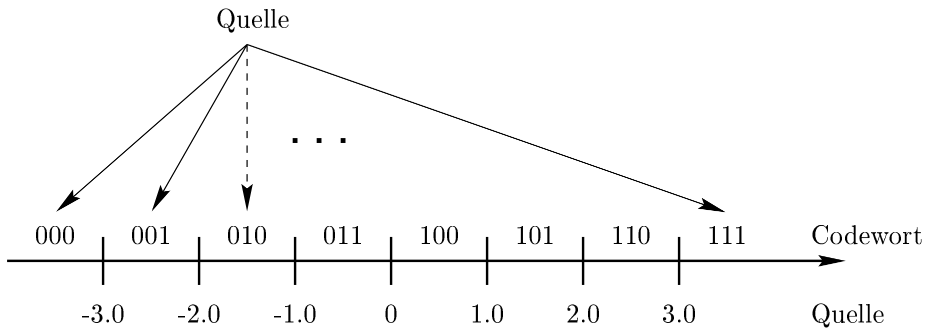


Abbildung 1: Eine 3-Bit-Quantisiervorschrift

7 Skalar- und Vektorquantisierung

7.1 Quantisierung

ist eine der einfachsten und allgemeinsten Ideen verlustbehafteter Komprimierung: Die Ausgabe einer Quelle muß durch eine endliche “kleine” Menge $\mathcal{C}$ von Codewörtern dargestellt werden, obwohl diese Ausgabe Werte aus einer sehr viel größeren, möglicherweise unendlichen Menge $\mathcal{A}$ annehmen kann. [2] Die Quantisiervorschrift liefert also eine o.E. surjektive Abbildung $Q_C : \mathcal{A} \to \mathcal{C}$.

Da die Güte der Codierung (Kompressionsrate bzw. Präzision) sowohl von Q_C als auch von der Decodiervorschrift $Q_D : \mathcal{C} \to \mathcal{A}$ abhängt, begreifen wir unter einem Quantisierer **Q** das Paar (Q_C, Q_D). Das Verhalten von **Q** läßt sich durch seine Ein/Ausgabefunktion

$$\mathcal{A} \to \mathcal{A}, \quad x \mapsto Q_D(Q_C(x))$$

veranschaulichen, die wir der Einfachheit halber auch mit **Q** bezeichnen.

Bild 1 zeigt eine 3-Bit-Quantisiervorschrift (d.h., es werden drei Bits für die Codewörter benötigt, so daß wir $\mathcal{C} = \{0,1\}^3$ annehmen können), an der zwei Dinge noch deutlicher werden:

1. Codewörter bezeichnen Intervalle.[3]

[2] Ist $\mathcal{A}$ unendlich, so handelt es sich bei der Quantisierung streng genommen nicht mehr um ein Teilgebiet der (rechnerbezogenen) Komprimierung, da Computer immer nur endliche Mengen darstellen können.

[3] Auch wenn es theoretisch möglich wäre, andere Arten von Mengen als Intervalle zu codieren, ist dies unter dem Blickwinkel der Fehlerminimierung nicht sinnvoll.

Codewort	Repräsentant
000	-3.5
001	-2.5
010	-1.5
011	-0.5
100	0.5
101	1.5
110	2.5
111	3.5

Tabelle 1: Die Decodiervorschrift Q_D

t	$4\cos(2\pi t)$	A/D Ausgabe	D/A Ausgabe	Abweichung
0.05	3.804	111	3.5	0.304
0.10	3.236	111	3.5	-0.264
0.15	2.351	110	2.5	-0.149
0.20	1.236	101	1.5	-0.264

Tabelle 2: Die Abtastung einer Cosinuswelle

2. Die Quantisiervorschrift $Q_C : \mathbb{R} \to \{0,1\}^3$ ist am Rand asymmetrisch, d.h., die zu codierenden Randintervalle sind meist größer als die inneren Intervalle.

Da dieses Q_C (natürlicherweise) nicht injektiv ist, läßt sich der Originalwert der Quelle aus solch einem Repräsentanten nicht eindeutig rekonstruieren, m.a.W., Q_C ist verlustbehaftet. Etwas allgemeiner sind A/D-Wandler als Quantisiervorschrift Q_C auffaßbar und entsprechend D/A-Wandler als Decodierer.

Die Decodiervorschrift Q_D wählt daher für jedes Intervall einen Repräsentanten aus, wie dies in Tabelle 1 für die 3-Bit-Quantisierung aufgelistet ist. Dort sind, von den Randintervallen abgesehen, die Repräsentanten die Mittelpunkte der codierten Intervalle. Kennt man die Quellenstatistik genauer, so ist es sinnvoll, andere Repräsentanten für die Intervalle zu wählen (s.u.). Für unseren 3-Bit-Quantisierer ist die Ein/Ausgabefunktion $\mathbf{Q}$ in Abb. 2 zu sehen.

Tabelle 2 zeigt die Wirkung dieser 3-Bit-Quantisierung bei Abtastung einer Cosinuswelle $4\cos(2\pi t)$ alle 50 ms. Dort ist auch die Abweichung des Repräsentanten vom ursprünglich zu codierenden Wert angegeben.

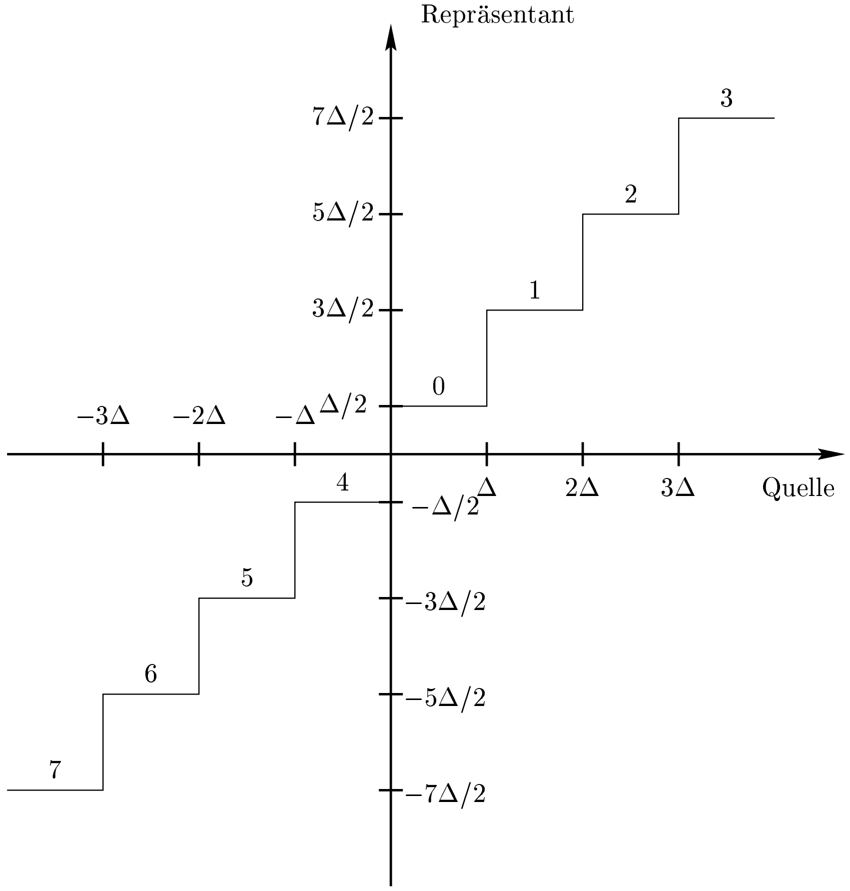


Abbildung 2: Die Ein/Ausgabefunktion eines 3-Bit-Quantisierers

Die Quantisierungsaufgabe besteht darin, einen Quantisierer $\mathbf{Q}$ so zu entwerfen, daß

- die Rate R (also die durchschnittliche Anzahl Bits, die für die Repräsentation eines Elementes aus $\mathcal{C}$ benötigt wird)[4] und
- den mittleren quadratischen Fehler σ_Q^2 (s. Kapitel 6)

zu minimieren. Da die Rate in unmittelbarem Zusammenhang zum Kompressionsquotienten steht, ist klar, daß diese beiden Forderungen einander widerstreitend sind. Geht man jedoch von Codewörtern fester gleicher Länge aus, so ist $R = \lceil \log_2 |\mathcal{C}| \rceil$ fixiert, so daß "nur" noch die Minimierung von σ_Q^2 verbleibt.

Es werde nun die Quellenstatistik durch eine Zufallsgröße X mit Wahrscheinlichkeitsdichte f_X modelliert. Geschieht die Quantisierung mit Hilfe von M Intervallen, ist also $|\mathcal{C}| = M$ mit $\mathcal{C} = \{1, \ldots M\}$, so haben wir die *Entscheidungsgrenzen* genannten Intervallendpunkte $b_0, \ldots, b_M$ sowie die Intervallrepräsentanten $y_1, \ldots, y_M$ anzugeben.[5] In unserer vorigen Notation ist daher für $1 \leq i \leq M$:

$$Q_C^{-1}(i) = (b_{i-1}, b_i],^6 \quad Q_D(i) = y_i \quad \text{und somit}$$

$$\mathbf{Q}(x) = y_i \text{ genau dann, wenn } x \in (b_{i-1}, b_i].$$

Daraus ergibt sich

$$\begin{aligned} \sigma_Q^2 &= \int_{-\infty}^{+\infty} (x - \mathbf{Q}(x))^2 f_X(x) dx \\ &= \sum_{i=1}^{M} \int_{b_{i-1}}^{b_i} (x - y_i)^2 f_X(x) dx \end{aligned} \tag{2}$$

Sind f_X und M vorgegeben und geht man von Codewörtern fester gleicher Länge aus, so besteht die Quantisierungsaufgabe darin, die Parameter $b_0, \ldots, b_M; y_1, \ldots y_M$ so zu wählen, daß σ_Q^2 gemäß Gleichung 2 minimiert wird.

Geht man nicht von Codewörtern fester gleicher Länge aus und codiert die (Indizes der) Intervallrepräsentanten z.B. gemäß dem Huffman-Verfahren, so beeinflußt diese Codierung gemeinsam mit der Wahl der Entscheidungsgrenzen die Rate des Quantisierers, die so zu einer von der Wahrscheinlichkeit $P(y_i)$ des Auftretens von y_i abhängigen Zufallsgröße wird.

[4]Komprimiert man z.B. 8-Bit-Wörter durch Fortlassen des niederwertigsten Bits, so ist $R = |\mathcal{C}| = 7$.

[5]$b_0 = -\infty$ und $b_M = +\infty$ seien zugelassen.

[6]Zu welchen Intervallen die Intervallgrenzen selbst zählen, ist unerheblich.

Wegen $P(y_i) = \int_{b_{i-1}}^{b_i} f_X(x)dx$ gilt

$$R = \sum_{i=1}^{M} \ell_i P(y_i) = \sum_{i=1}^{M} \ell_i \int_{b_{i-1}}^{b_i} f_X(x)dx, \tag{3}$$

wobei ℓ_i die Länge des y_i entsprechenden Codewortes ist.

So sind also die Aufgaben, optimale Entscheidungsgrenzen, Repräsentanten sowie Codierungen zu finden, miteinander verwoben. Im allgemeinen stellen sich daher zwei Arten von Optimierungsaufgaben:

1. Ist eine Verzerrungsschranke $D^* \geq \sigma_Q^2$ vorgegeben, so ist die Rate gemäß Gleichung 3 zu minimieren.

2. Ist eine Ratenschranke $R^* \geq R$ vorgegeben, so ist die Verzerrung nach Gleichung 2 zu minimieren.

Da die Entropie des Quantisierers durch

$$H = -\sum_{i=1}^{M} P(y_i)\log_2(P(y_i)) \tag{4}$$

gegeben ist, kann man auch noch eine dritte Quantisieraufgabe stellen:

3. Ist eine Entropieschranke $H^* \geq H$ vorgegeben, so ist die Verzerrung nach Gleichung 2 zu minimieren.

Diese Minimierungsaufgaben werden leicht nicht-trivial. Im folgenden kümmern wir uns –sofern nicht anders vermerkt– um die zweite der Minimierungsaufgaben, und wir gehen von einer Codierung mit fester gleicher Wortlänge aus.

7.2 Gleichquantisierer

Wir wollen im folgenden den in der Praxis durchaus üblichen *Gleichquantisierer* untersuchen, bei dem alle Intervalle (mit Ausnahme evtl. der äußeren Intervalle) gleich groß sind und als Intervallrepräsentanten die Mittelpunkte dieser (inneren) Intervalle genommen werden, womit sowohl die Entscheidungsgrenzen als auch die Repräsentanten jeweils gleichabständig liegen; dieser Abstand wird auch *Schrittweite* genannt und mit Δ bezeichnet. Es geht jetzt um den Entwurf von verzerrungsminimierenden Gleichquantisierern bei vorgegebener Quellenverteilung und vorgegebenem Umfang M von $|\mathcal{C}|$.

Wir wollen jetzt einen Gleichquantisierer für eine im Intervall $[-X_\mu, X_\mu]$ gleichverteilte Quelle entwerfen mit Hilfe von M Intervallen, wobei M im folgenden o.E. gerade sei. Damit gilt:

$$\Delta = \frac{2X_\mu}{M}, \qquad f_X(x) = \begin{cases} \frac{1}{2X_\mu}, & x \in [-X_\mu, X_\mu] \\ 0, \text{sonst} \end{cases}$$

$$\sigma_Q^2 = 2\sum_{i=1}^{M/2} \int_{(i-1)\Delta}^{i\Delta} (x - \frac{2i-1}{2}\Delta)^2 \frac{1}{2X_\mu} dx = \ldots = \frac{\Delta^2}{12}$$

Nun berechnen wir das Verhältnis von Signal zur Verzerrung. Da die Wahrscheinlichkeitsdichte der Quelle symmetrisch zum Ursprung ist, gilt $E[X] = 0$, d.h., der mittlere quadratische Wert der Quelle $E[X^2]$ ist gleich der Varianz, also ist $\sigma_X^2 = \frac{(2X_\mu)^2}{12}$. Daher ist:

$$\begin{aligned} SNR(dB) &= 10\log_{10}(\frac{\sigma_X^2}{\sigma_Q^2}) \\ &= 10\log_{10}(\frac{(2X_\mu)^2/12}{(\frac{2X_\mu}{M})^2/12}) \\ &= 6.02\log_2 M dB \end{aligned}$$

Dies ist sozusagen der Präzisionsgewinn pro Bit, wie das am Sena-Bild 8.7 gut zu sehen ist.

Die Annahme der Gleichverteilung ist tatsächlich meist falsch (Bilder haben meist Häufungen bei gewissen Grauwerten.), ebenso die der Unabhängigkeit aufeinanderfolgender Ereignisse (Bei Bildern sind benachbarte Pixel häufig gleich.) Die Methode der Vektorquantisierung (s. Abschnitt 7.5) geht auf den letzteren Punkt ein. Zunächst wollen wir uns mit dem Problem befassen, einen Gleichquantisierer für eine Quelle X mit beliebiger, bezüglich des Ursprungs symmetrischer Wahrscheinlichkeitsdichte f_X zu entwerfen. Für gerade M ist zu minimieren:

$$\begin{aligned} \sigma_Q^2 &= 2\sum_{i-1}^{M/2-1} \int_{(i-1)\Delta}^{i\Delta} \left(x - \frac{2i-1}{2}\Delta\right)^2 f_X(x)dx \\ &+ 2\int_{M/2-1}^{\infty} \left(x - \frac{M-1}{2}\Delta\right)^2 f_X(x)dx, \end{aligned}$$

was durch Nullsetzen der Ableitung $\frac{d\sigma_Q^2}{d\Delta}$ geschehen kann.

Auf diese Weise kann man die in Tabelle 3 aufgeführten optimalen Schrittweiten errechnen unter der Normierung $\sigma_X^2 = 1$ für drei Verteilungen. Die angegebene Schrittweite für die Gleichverteilung ergibt sich aus der Normierungsbedingung $\sigma_X^2 = (2X_\mu)^2/12 = 1$. Es nimmt nicht wunder, daß der

Alph.-größe	Gleichverteilung Schrittweite	SNR	Gaußverteilung Schrittweite	SNR	Laplaceverteilung Schrittweite	SNR
2	1.732	6.02	1.596	4.40	1.414	3.00
4	0.866	12.04	0.9957	9.24	1.0873	7.05
6	0.577	15.58	0.7334	12.18	0.8707	9.56
8	0.433	18.06	0.5860	14.27	0.7309	11.39
10	0.346	20.02	0.4908	15.90	0.6334	12.81
12	0.289	21.60	0.4238	17.25	0.5613	13.98
14	0.247	22.94	0.3739	18.37	0.5055	14.98
16	0.217	24.08	0.3352	19.36	0.4609	15.84
32	0.108	30.10	0.1881	24.56	0.2799	20.46

Tabelle 3: Optimale Schrittweiten für Gleichquantisierer

Eingabeverteilung	Gleichverteilung	Gaußverteilung	Laplaceverteilung
Gleich-	18.06	15.56	13.29
Gauß-	12.40	14.27	13.37
Laplace-	8.80	10.79	11.39

Tabelle 4: Der Einfluß von Verteilungsannahmen auf SNR

Präzisionsgewinn pro Bit bei der Laplaceverteilung geringer ist als bei der Gaußverteilung und bei der Gaußverteilung wiederum geringer ist als bei der Gleichverteilung: schließlich "entspricht" die Gleichverteilung dem Gleichquantisierer, und die Laplace-Verteilung ist "am wenigsten gleichverteilt".

Tabelle 4 zeigt, daß es nicht unwichtig ist, welche Verteilung für die Quelle angenommen wurde; ist (aufgrund einer falschen Verteilungsannahme) die Schrittweite kleiner als die optimale, so ist der Fehler größer als wenn die Schrittweite überschätzt wird. Durch statistische Testverfahren, die außerhalb unserer Vorlesung liegen, kann man diesem Fehler begegnen. Ähnlich kann man vorgehen, wenn die angenommene Varianz falsch geschätzt wurde.[7]

7.3 Adaptive Quantisierung

Um den genannten Entwurfsfehlern (Annahme einer falschen Verteilung bzw. Varianz) grundsätzlich zu begegnen, betrachten wir jetzt die *adaptive Quantisierung*. Hierbei nehmen wir an, die Verteilung der Quelle ändere

[7] Eine elementare Einführung finden Sie in [1].

sich mit der Zeit (Dies ist sehr typisch bei Sprachsignalen zu beobachten.) oder aber ist uns schlicht unbekannt. Ändert sich lediglich der Mittelwert, so ist eine Form der Differentialcodierung zu wählen, die im nächsten Kapitel behandelt wird. Andernfalls empfiehlt sich eine Anpassung der Parameter des Quantisierers an die Quellenstatistik. Hier gibt es zwei Grundstrategien:

1. Voradaptierung und

2. Rückadaptierung.

7.3.1 Voradaptierung

ist ein typisches "off-line"-Verfahren. Die Ausgabe der Quelle wird in Blöcke unterteilt, deren Statistik separat analysiert wird; gemäß dieser Analyse werden die Quantisierungsparameter gesetzt. Wie jedes "off-line"-Verfahren bringt auch dieses eine Verzögerung der Übertragung durch den Puffer-Effekt mit sich; diese Verzögerung ist umso größer, je länger der Puffer ist. Andererseits erlaubt ein langer Puffer eine gute Verteilungsanalyse. Wie ist also der Puffer zu wählen? Da die Analyse auf den dem Decodierer nicht zur Verfügung stehenden Eingabedaten beruht, müssen die so ermitteltеten Parameter als Begleitinformationen mit übertragen werden.

Wie Bild 8.15 im Vergleich zu Bild 8.7 unten rechts zeigt, kann diese Strategie durchaus Beachtliches leisten: Das in 8×8-Blöcke unterteilte Sena-Bild wurde wiederum unter Annahme der Gleichverteilung durch einen Gleichquantisierer komprimiert, nur daß jetzt als Begleitinformation je Block die Minimal- und Maximalwerte der Gleichverteilung mitgesendet wurden, was eine Übertragung von 3,25 Bits pro Pixel bedeutet anstelle von 3 Bits pro Pixel in Bild 8.7.

7.3.2 Rückadaptierung

Nur die in der Vergangenheit quantisierten Proben können zur Adaptierung verwendet werden. Will man die Übertragung von Begleitinformationen vermeiden, so besteht die Schwierigkeit darin, daß die unquantisierte Eingabe nur dem Codierer bekannt ist und so nicht zur Veränderung der Quantisiererparameter verwendet werden kann. Eine einfache Lösung für dies Problem wurde von N.S. Jayant gefunden; Grundgedanke hierbei ist: Fällt die letzte Eingabe in ein inneres Intervall, so schrumpft die Schrittweite, fällt sie in ein äußeres Intervall, so wird die Schrittweite größer. Trivialerweise steht die Information, in welches Intervall die Eingabe fiel, auch dem Decodierer zur Verfügung, so daß keine Begleitinformationen versendet werden müssen.

n	Δ_n	Eingabe	Code	Repräsentant	Abweichung	nächste Schrittweite
0	0.5	0.1	0	0.25	0.15	$\Delta_1 = M_0 \times \Delta_0$
1	0.4	-0.2	4	-0.2	0.0	$\Delta_2 = M_4 \times \Delta_1$
2	0.32	0.2	0	0.16	0.04	$\Delta_3 = M_0 \times \Delta_2$
3	0.256	0.1	0	0.128	0.028	$\Delta_4 = M_0 \times \Delta_3$
4	0.2048	-0.3	5	-0.3072	-0.0072	$\Delta_5 = M_5 \times \Delta_4$
5	0.1843	0.1	0	0.0922	-0.0078	$\Delta_6 = M_0 \times \Delta_5$
6	0.1475	0.2	1	0.2212	0.0212	$\Delta_7 = M_1 \times \Delta_6$
7	0.1328	0.5	3	0.4646	-0.0354	$\Delta_8 = M_3 \times \Delta_7$
8	0.1594	0.9	3	0.5578	-0.3422	$\Delta_9 = M_3 \times \Delta_8$
9	0.1913	1.5	3	0.6696	-0.8304	$\Delta_{10} = M_3 \times \Delta_9$
10	0.2296	1.0	3	0.8036	0.1964	$\Delta_{11} = M_3 \times \Delta_{10}$
11	0.2755	0.9	3	0.9643	0.0643	$\Delta_{12} = M_3 \times \Delta_{11}$

Tabelle 5: Zur Arbeitsweise eines Jayant-Quantisierers

Genauer arbeitet der *Jayant-Quantisierer* wie folgt: Jedem der M Intervalle wird ein Faktor M_i, $1 \leq i \leq M$, zugeordnet. Die Faktoren der inneren Intervalle sind kleiner als Eins, die der äußeren größer als Eins. Bezeichnet Δ_n die Schrittweite zur Verarbeitung der n-ten Eingabe und falle diese ins Intervall $\ell(n)$, so wird die Schrittweite gemäß der Vorschrift

$$\Delta_{n+1} = M_{\ell(n)}\Delta_n$$

aktualisiert.

Tabelle 5 mag die Arbeitsweise eines Jayant-Quantisierers erläuern helfen. Ausgehend von dem Quantisierer aus Abb. 2 mit den dort angegebenen Intervallcodierungen sowie den Faktoren $M_0 = M_4 = 0,8$, $M_1 = M_5 = 0,9$, $M_2 = M_6 = 1$, $M_3 = M_7 = 1,2$ und der Anfangsschrittweite $\Delta_0 = 0,5$ erhält man die angegebene Schrittweiten- und Ausgabenfolge. Man berechne einmal den so erhaltenen quadratischen Fehler und vergleiche ihn mit einem nicht-adaptiven Quantisierer! Um aus numerischen Gründen zu vermeiden, daß Δ_n "zu nahe" an Null an Null kommt bzw. "zu groß" wird, werden häufig Grenzwerte Δ_{min} bzw. Δ_{max} definiert.

Wie wählt man die Faktoren M_i? Als Faustregel kann gelten: Je mehr sich die Faktoren von Eins unterscheiden, desto rascher reagiert der Jayant-Quantisierer. Eine zu rasche Reaktion kann allerdings zu Instabilitäten führen. Geht man von einem stationären Eingabeprozeß aus, bei dem schließlich die Wahrscheinlichkeiten, mit eine einzelnen Eingabe im i-ten Intervall zu landen, P_i beträgt, so sollten die P_i der Gleichung

$$\prod_{i=1}^{M} M_i^{P_i} = 1$$

genügen.

Abschließend noch zwei Warnungen zu adaptiven Verfahren:

- Kennt man die Quellenstatistik genau, so ist ein diesbeüglich optimierter Quantisierer einem adaptiven überlegen. Ganz allgemein ist es bei der Datenkompression essentiell, alle verfügbaren Informationen über die Quelle auszunutzen.
- Wählt man die Adaptionsparameter "ungeschickt" im Hinblick auf die Eingabedaten, so kann es –wie ja allgemein von Reglern bekannt– zu Schwingeffekten kommen.

7.4 Allgemeine Quantisierer

Ist die Quelle nicht gleichverteilt, so erscheint es sinnvoll, die Intervallaufteilung nicht gleichmäßig wie beim Gleichquantisierer zu gestalten. In diesem Abschnitt wollen wir solche allgemeinen Quantisierer betrachten.

Eine Idee ist es, bei bekannter Wahrscheinlichkeitsdichte f_X der Quelle direkt zu versuchen, Ausdruck (2) zu minimieren. Dies führt auf ein wechselseitig rekursives, nicht-lineares Gleichungssystem für die Repräsentanten und Entscheidungsgrenzen, deren Lösung –fußend auf Ideen der Variationsrechnung– unabhängig von einander S.P. Lloyd, J. Max sowie J. Lukaszewicz und H. Steinhaus in den 50er Jahren angaben. Da Varianten hiervon noch später von Interesse sein werden, geben wir das Verfahren in Tab. 6 an. Zentroide sind eine die Wahrscheinlichkeitsdichte berücksichtigende Verallgemeinerung des Mittelwertbegriffs, wie man sieht, wenn man die Gleichverteilung als Dichte ansetzt; sie lassen sich als normierte Erwartungswerte begreifen. Evtl. ist es sinnvoll, andere als die äußeren Entscheidungsgrenzen als fix anzunehmen und dann den Algorithmus geeignet abzuwandeln.

Eine andere Idee ist es, einem Gleichquantisierer eine Kompressor c genannte Transformation vorzuschalten und eine Expander e genannte Transformation nachzuschalten. Ein Beispiel für solche sog. *zusammengesetzten Quantisierer* liefern der Kompressor

$$c(x) = \begin{cases} 2x, & \text{falls } x \in [-1,1] \\ \frac{2x}{3} + \frac{4}{3}, & \text{falls } x \in [1,+\infty) \\ \frac{2x}{3} - \frac{4}{3}, & \text{falls } x \in (-\infty,-1] \end{cases}$$

und der Expander $e = c^{-1}$. In diesem Fall ist die Schrittweite des Quantisierers Q_C im Intervall $[-1,1]$ kleiner als außerhalb dieses Intervalls, was

1. Wähle Anfangsrepräsentanten $\{y_i^{(0)}\}_{i=1}^{M}$ und Abbruchschwellwert ε für die Verzerrungsdifferenz.
   Setze $D^{(-1)} = 0,\ k = 0$.

2. Berechne neue innere Entscheidungsgrenzen $b_1^{(k)}, \ldots, b_{M-1}^{(k)}$ (und damit die Intervalle) mit:
$$b_j^{(k)} = \frac{y_{j+1}^{(k)} + y_j^{(k)}}{2}\,.$$

3. Berechne die neue Verzerrung
$$D^{(k)} = \sum_{i=1}^{M} \int_{b_{i-1}^{(k)}}^{b_i^{(k)}} (x - y_i)^2 f_X(x)dx\,.$$

4. Falls $\left|D^{(k)} - D^{(k-1)}\right| < \varepsilon$, STOP.

5. Sonst: Setze $k = k+1$ und berechne neue Repräsentanten als Zentroide:
$$y_j^{(k)} = \frac{\int_{b_{j-1}^{(k-1)}}^{b_j^{(k-1)}} x f_X(x)dx}{\int_{b_{j-1}^{(k-1)}}^{b_j^{(k-1)}} f_X(x)dx}\,.$$

   Gehe zu Schritt 2.

Tabelle 6: Das Lloyd et. al.-Verfahren

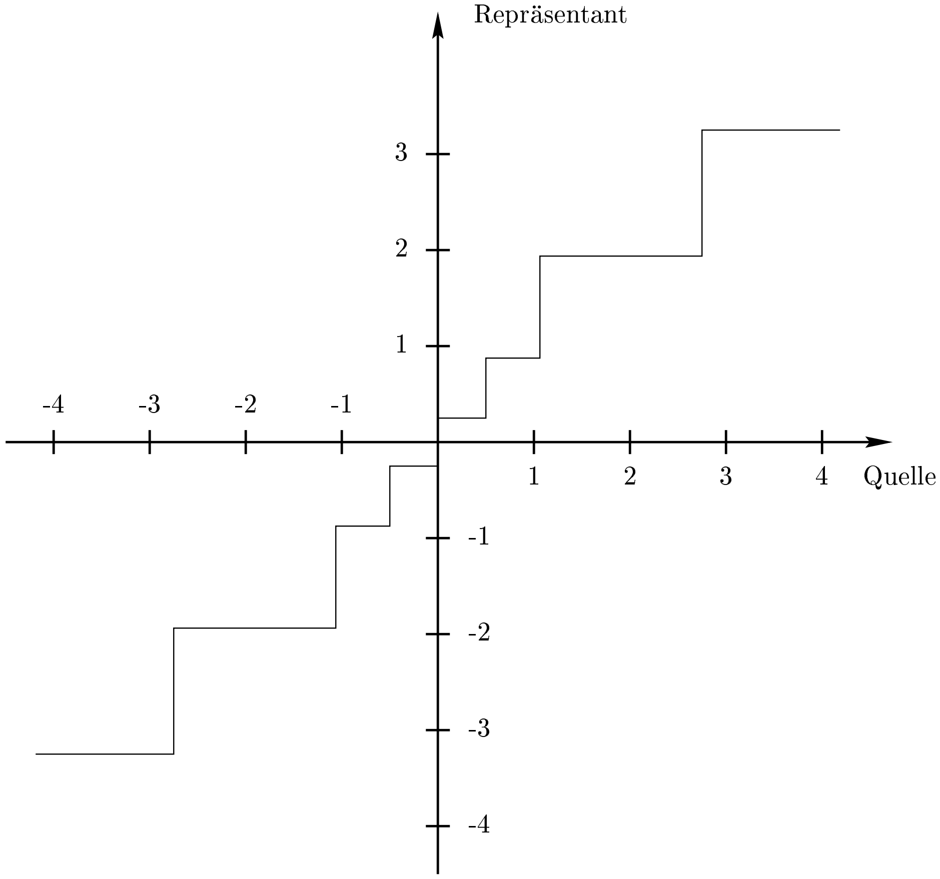


Abbildung 3: Die Ein/Ausgabefunktion eines 3-Bit-Quantisierers

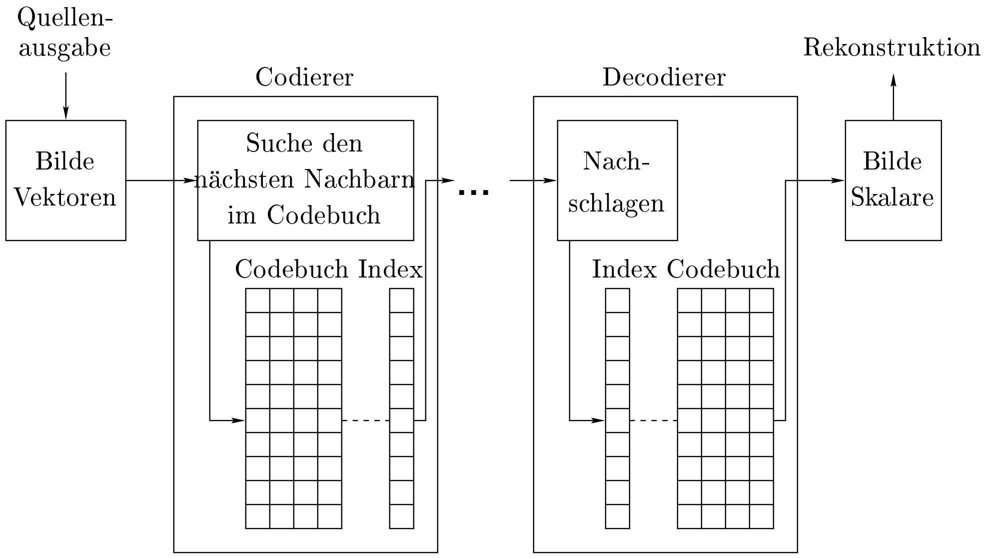


Abbildung 4: Die Arbeitsweise eines Vektorquantisierers

z.B. bei Gauß- oder Laplace-verteilten Quellen erwünscht ist. Bei "Zwischenschalten" eines 3-Bit-Gleichquantisierers liefert der so definierte zusammengesetzte Quantisierer das in Abb. 3 gezeigte Verhalten **Q**. Die von Telefongesellschaften eingesetzten Quantisierer sind als zusammengesetzte definiert.

7.5 Skalare Quantisierung versus Vektorquantisierung

Bislang hatten wir versucht, einzelne Quellenausgaben (also skalare Werte) zu quantisieren. Man spricht hierbei auch von der *skalaren Quantisierung.* Tatsächlich bestehen jedoch häufig räumliche (Bsp.: Bilder) oder zeitliche (Bsp.: Sprache) Abhängigkeiten zwischen aufeinanderfolgenden Quellenausgaben. Ein wenig wurde dies bereits bei der adaptiven Quantisierung berücksichtigt.

Bei der jetzt zu betrachtenden *Vektorquantisierung* werden immer L aufeinanderfolgende Quellenausgaben zu einem L-dimensionalen Vektor zusammengefaßt und gemeinsam quantisiert. Der Codierer enthält im wesentlichen eine Codebuch genannte indizierte Tabelle sog. Codevektoren und arbeitet für jeden Eingabevektor $\vec{i}$ wie folgt:

1. Suche Codevektor $\vec{v}$, der der Eingabe $\vec{i}$ am nächsten kommt.

2. Übertrage den so gefundenen Index von $\vec{v}$.

Der Decodierer enthält dasselbe Codebuch und kann über den Index leicht auf den $\vec{i}$ quantisierenden Codevektor $\vec{v}$ zugreifen.

Das in Bild 4 nochmals veranschaulichte Schema eines Vektorquantisierers ist insofern asymmetrisch, als daß der Codierer einen erheblichen Rechenaufwand zu erledigen hat (Codebücher können sehr groß sein, und die Bestimmung des nächsten Nachbarn [s. Vorlesung "Algorithmische Geometrie"] kann so viel Zeit in Anspruch nehmen.), während die Decodierung geradezu trivial ist. Dieses allgemeine Schema der Vektorquantisierung ist daher insbesondere für Anwendungen geeignet, wo ein einmaliger Aufwand in der Codierung vielfach durch einfache Decodierung genutzt wird, z.B. im Multimediabereich. Aufgrund der sehr geringen Raten (d.h., hohen Kompressionsquotienten) ist Vektorquantisierung aber überhaupt sehr populär, insbesondere auch bei der Sprachübertragung.

Besteht ein Codebuch aus K L-dimensionalen Vektoren, so werden (bei Annahme fester Wortlänge der übertragenden Indizes) je Skalar lediglich $\lceil \log_2 K \rceil / L$ Bits benötigt, was sehr große Codebücher gestattet.

Veranschaulichen wir uns die Vektorquantisierung an zwei Beispielen.

Beispiel 1: Länge und Gewicht von Personen.

Selbst wenn –wie in Bild 5 dargestellt– Länge bzw. Gewicht[8] einer Stichprobe getrennt betrachtet jeweils gleichverteilt sein mögen in einem Intervall I_ℓ bzw. I_g, so darf man nicht die im Bild dargestellte Gleichverteilung im Quadrat $I_\ell \times I_g$ erwarten, denn Länge und Gewicht von Personen sind korreliert, so wie dies in Bild 6 angedeutet ist. Eine zweidimensionale Vektorquantisierung kann hier helfen.

Beispiel 2: Diskussion einer Laplace-verteilten Quelle

Wie aus Tabelle 3 ersichtlich, besitzt ein optimaler 3-Bit-Gleichquantisierer für eine Laplace-verteilte Quelle eine Schrittweite von $\Delta = 0,7309$. Hier gilt aber: $P(x \in [0, \Delta)) = 0,3242$ und $P(x \in [3\Delta, \infty)) = 0,0225$. Im vorigen Beispiel war die Wahl eines Vektorquantisierers durch die natürliche Korrelation der Eingabedaten begründet. Hier wollen wir –ganz im Gegensatz dazu– annehmen, aufeinanderfolgende Werte der Quelle seien unabhängig. Fassen wir dennoch die Eingabewerte zu Paaren zusammen, so entspricht

[8] Uns ist bekannt, daß das Kilogramm die Einheit der Masse und nicht des Gewichts ist; wir passen uns hier lediglich dem umgangssprachlichen Gebrauch an. Personenwaagen messen tatsächlich das Gewicht und rechnen dies dann unter Berücksichtigung der Gravitation in Kilogramm um.

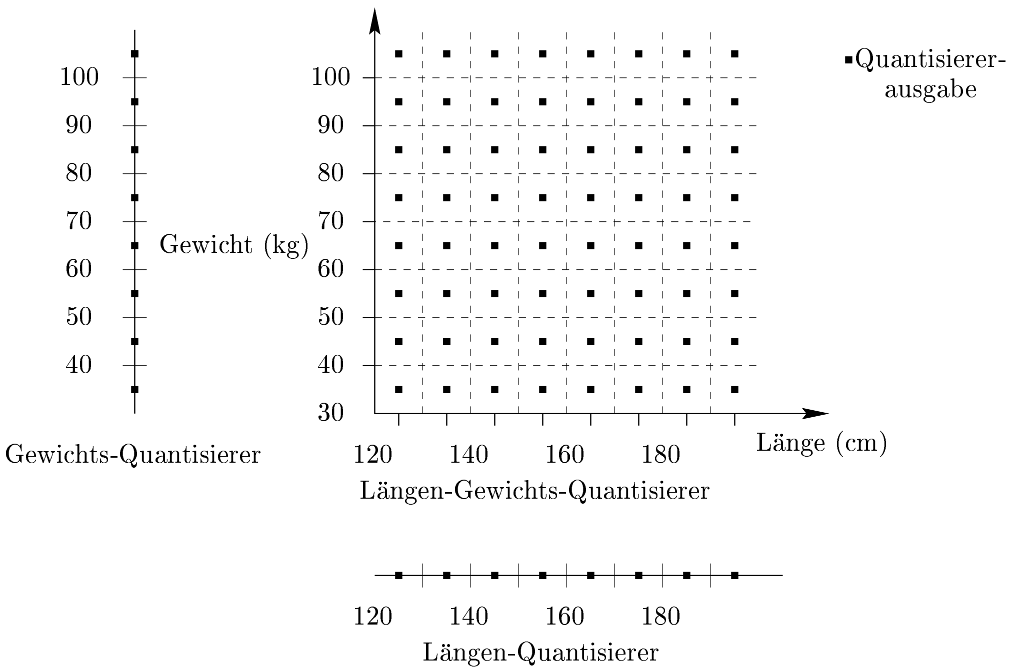


Abbildung 5: Länge und Gewicht von Personen

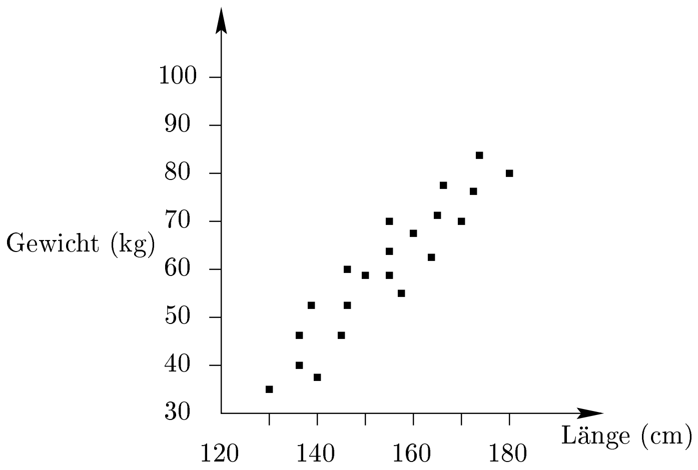


Abbildung 6: Eine Stichprobe von Länge und Gewicht von Personen

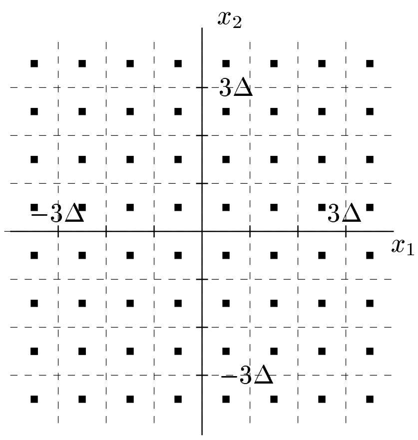


Abbildung 7: Die Gitterstruktur eines gepaarten 3-Bit-Gleichquantisierers

dem (skalaren) Gleichquantisierer ein Vektorquantisierer, bei dem die "Einzugsbereiche" eines Vektors eine gitterartige Struktur liefern, s. Bild 7.

Die Wahrscheinlichkeit, daß ein Eingabepaar in das rechte obere "Quadrat" fällt, beträgt (bei Annahme der Unabhängigkeit) lediglich

$$P((x,y) \in [3\Delta,\infty)\times[3\Delta,\infty)) = P(x \in [3\Delta,\infty))\cdot P(y \in [3\Delta,\infty)) = 0,0005\,.$$

Daher liegt die Idee nahe, den rechten oberen Codevektor durch den Nullpunkt zu ersetzen, wodurch sich die in Bild 8 angedeutete Gitterstruktur ergibt. Das Verhältnis SNR steigt dadurch von $11,44dB$ auf $11,73dB$.

7.6 Der Entwurf guter Codebücher

Wie in obigem Beispiel 1 kann die innere Struktur der Eingabe zu einem optimierten Codebuch-Entwurf genutzt werden. Das Auffinden von Clustern muß bei großen Codebüchern automatisiert werden. Dazu dient eine von Lindo, Buzo und Gray ersonnene Modifikation des Lloydschen Algorithmus (vgl. Tab. 6). Anstelle der Entscheidungsgrenzen treten die Quantisierungsregionen, und die Verzerrung bzw. Zentroide sind entsprechend komplizierte Integrale, s. Tab. 7.

Aufgrund der Kompliziertheit der auftretenden Integrale gibt es auch eine vereinfachte, sozusagen diskretisierte Version, bei der mit einer Trainingsmenge $\{X_n\}_{n=1}^{N}$ gearbeitet wird. Dieses Verfahren ist in Tab. 8 beschrieben. Es kommt ohne Annahmen über die Quellenstatistik aus.

Als Beispiel jetzt die in Tab. 9 dargestellte Trainingsmenge, wobei wir zunächst mit dem in Tab. 10 gegebenen Codebuch anfangen wollen. Diesem

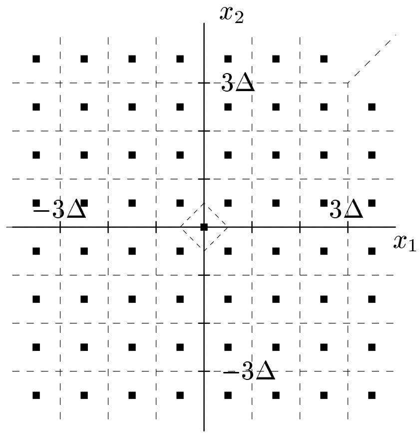


Abbildung 8: 64-elementiges Codebuch eines 2-dimensionalen Vektorquantisierers

1. Wähle Anfangsrepräsentanten $\{Y_i^{(0)}\}_{i=1}^{M}$ und Abbruchschwellwert ε für die Verzerrungsdifferenz.
   Setze $D^{(-1)} = 0$, $k = 0$.

2. Berechne neue Quantisierungsregionen mit:
$$V_j^{(k)} = \{X \mid \forall i \neq j : d(X, Y_j) < d(X, Y_i)\}\,.$$

3. Berechne die neue Verzerrung
$$D^{(k)} = \sum_{i=1}^{M} \int_{V_i^{(k)}} \left\| \vec{x} - Y_i^{(k)} \right\|^2 f_X(\vec{x}) d\vec{x}\,.$$

4. Falls $\frac{|D^{(k)} - D^{(k-1)}|}{D^{(k)}} < \varepsilon$, STOP.

5. Sonst: Setze $k = k + 1$ und berechne neue Repräsentanten als Zentroide:
$$Y_j^{(k)} = \frac{\int_{V_j^{(k-1)}} \vec{x} f_X(\vec{x}) d\vec{x}}{\int_{V_j^{(k-1)}} f_X(\vec{x}) d\vec{x}}\,.$$
   Gehe zu Schritt 2.

Tabelle 7: Das Lindo-Buzo-Gray-Verfahren I

Es steht eine Trainingsmenge $\{X_n\}_{n=1}^{N}$ zur Verfügung.

1. Wähle Anfangsrepräsentanten $\{Y_i^{(0)}\}_{i=1}^{M}$ und Abbruchschwellwert ε für die Verzerrungsdifferenz.
   Setze $D^{(-1)} = 0$, $k = 0$.

2. Berechne neue Quantisierungsregionen mit:

$$V_j^{(k)} = \{X_n \mid \forall i \neq j : d(X_n, Y_j) < d(X_n, Y_i)\}\,.$$

   Sollte ein $V_j^{(k)}$ leer sein, so teile man statt dessen die größte Region in zwei "Hälften".

3. Berechne die neue Verzerrung

$$D^{(k)} = \sum_{i=1}^{M} \sum_{X_n \in V_i^{(k)}} \|X_n - Y_i\|^2\,.$$

4. Falls $\frac{\left|D^{(k)} - D^{(k-1)}\right|}{D^{(k)}} < \varepsilon$, STOP.

5. Sonst: Setze $k = k + 1$ und berechne neue Repräsentanten als Mittelwerte:

$$Y_j^{(k)} = \frac{\sum_{X_n \in V_j^{(k)}} X_n}{\left|\{X_n \in V_j^{(k)}\}\right|}\,.$$

   Gehe zu Schritt 2.

Tabelle 8: Das Lindo-Buzo-Gray-Verfahren II

Länge	Gewicht
180	90
180	88
163	60
110	20
148	60
155	57
160	75
150	55
163	81
140	46
143	44
175	86

Tabelle 9: Trainingsmenge für den Codebuchentwurf

Länge	Gewicht
113	25
113	59
188	59
200	90

Tabelle 10: Anfangscodebuch

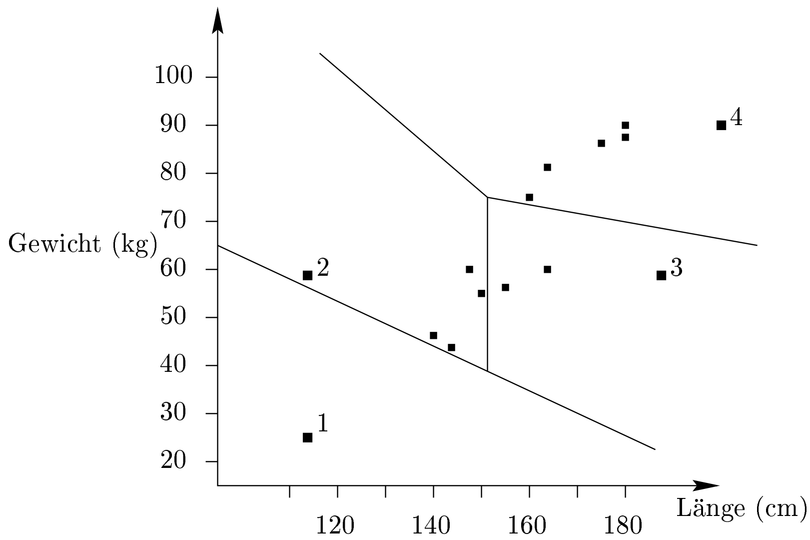


Abbildung 9: Anfängliche Quantisierungsregionen

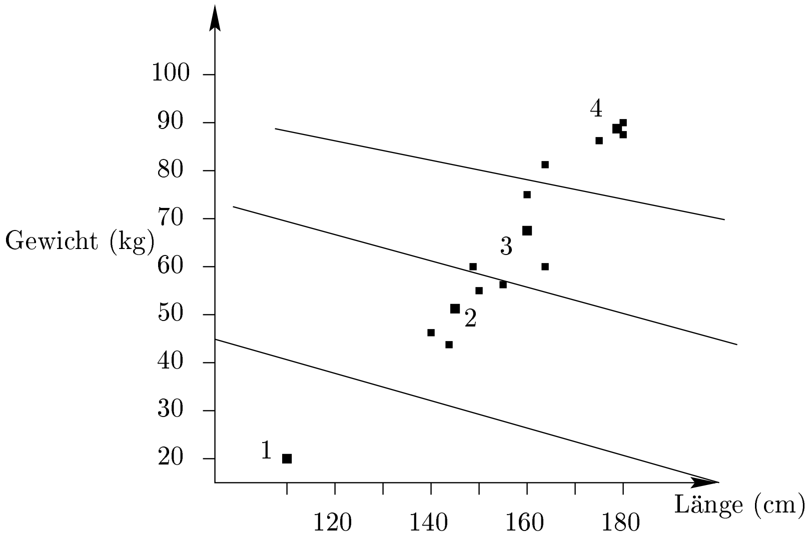


Abbildung 10: Regionen nach einem Durchlauf des LBG-Verfahrens

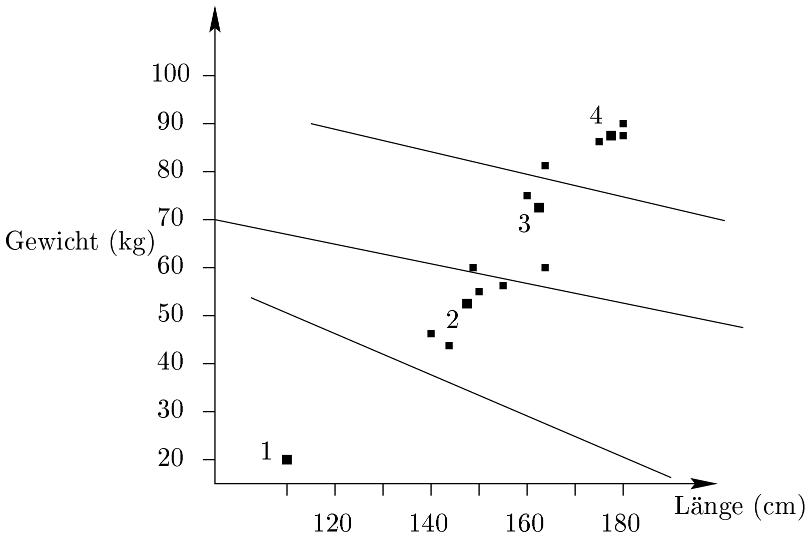


Abbildung 11: Regionen nach zwei Durchläufen des LBG-Verfahrens

Länge	Gewicht
188	25
188	64
188	59
200	90

Tabelle 11: Ein anderes Anfangscodebuch...

Länge	Gewicht
110	20
161,5	78
149,8	53,7
178,3	88

Tabelle 12: ...führt zu einem anderen Endcodebuch...

Codebuch entsprechen die in Bild 9 gezeigten Quantisierungsregionen. Nach einer Iteration bzw. zwei Iterationen des LBG-Verfahrens ergeben sich die in Bild 10 bzw. Bild 11 gezeigten Regionen.

Leider kann die Güte des so erhaltenen Codebuchs stark von den Anfangsrepräsentanten abhängen: Beginnt man beispielsweise mit dem Codebuch aus Tab. 11, so erhält man schließlich als Codebuch 12, was den in Bild 12 dargestellten Regionen entspricht.

Als Abhilfe schlagen Lindo, Buzo und Gray die in Tab. 13 angegebene Aufspalttechnik vor (der Abbruchschwellwert ε und die Vektorstörung δ sowie die Trainingsmenge sind als vorweg gegebene Parameter aufzufassen), vgl. auch Tab. 14 sowie die dazu gehörigen Regionen in den Bildern 13 und14 für das Längen/Gewichtsbeispiel. Eine quasi entgegengesetzte Idee zum

Entwirf K Bit großes Anfangscodebuch:

1. Falls $K = 0$: Nimm Mittelwert der Trainingsmenge $\{X_n\}_{n=1}^N$ als Codevektor.
2. Falls $K > 0$: Entwirf (rekursiv) $K - 1$ Bit großes Anfangscodebuch; Initialisiere K-Bit Codebuch durch 2^{K-1} sow erhaltene Vektoren sowie um fixes δ gestörte Vektoren;
   Starte LBG-Algorithmus zur Ermittlung eines K-Bit-Anfangscodebuchs.

Tabelle 13: Das Lindo-Buzo-Gray-Verfahren III

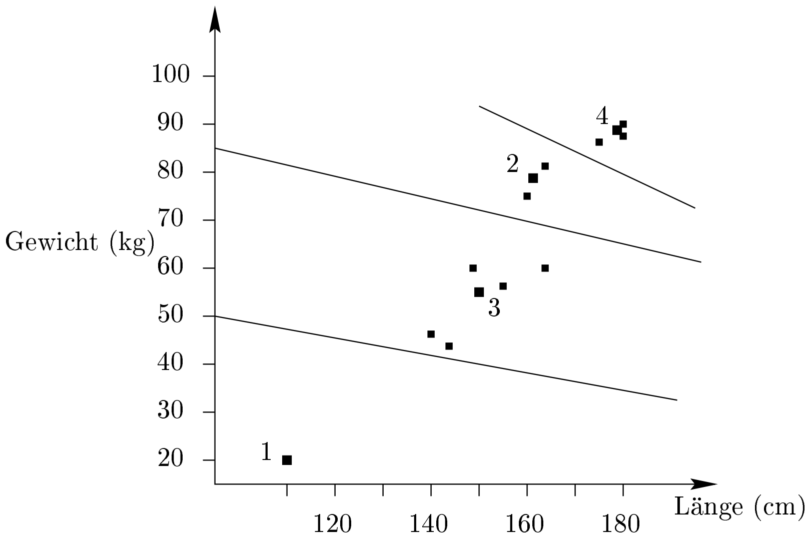


Abbildung 12: ... mit den angegebenen Regionen.

Entwurf des Anfangscodebuchs hatte Equitz: Ausgehend von einer Trainingsmenge werden solange paarweise nächstbenachbarte Cluster vereinigt, angefangen mit der Clustermenge $\{\{X_n\}\}_{n=1}^{N}$, bis der erwünschte Codebuchumfang erreicht ist; dabei ist der Abstand zwischen Clustern derjenige ihrer Mittelwerte. Häufig am einfachsten ist es, das Anfangscodebuch durch Würfeln als Teilmenge der Trainingsmenge zu wählen. Man kann dann den LBG-Algorithmus auf mehreren solchen Anfangscodebüchern laufen lassen und schließlich das beste Codebuch sich heraussuchen.

Zwei grundsätzliche Probleme birgt das LBG-Verfahren noch in sich:

1. Woher bekommt man "gute" Trainingsmengen für eine Anwendung?

   Bilder haben häufig unterschiedliche Charakteristika; ein auf ein Bild optimiertes Codebuch mag für ein anderes Bild nur bedingt brauchbar sein. Umgekehrt mindert ein als Begleitinformation übertragenes komplettes Codebuch natürlich die (bei Vektorquantisierern ansonsten hervorragende) Kompressionsrate, s. Tab. 15. Diesen Einfluß kann man gut an Tab. 16 beobachten, wo der Overhead für das einmalige (unkomprimierte) Übertragen des verwendeten Codebuchs zu sehen ist.

2. Das Auffinden des nächsten Codevektors kostet bei großen Codebüchern viel Zeit.

Das erste Problem kann gemildert werden, wenn man die Mittelwerte jedes Blocks ("Vektors") skalarquantisiert überträgt und lediglich die Abweichun-

Codebuch	Länge	Gewicht
0 Bit	155	64
1 Bit Anfangs-	155	64
	180	69
1 Bit End-	145	49
	173	84
2 Bit Anfangs-	145	49
	171	54
	173	84
	198	89
2 Bit End-	130	37
	155	58
	163	78
	178	88

Tabelle 14: Die Aufspalttechnik am Beispiel

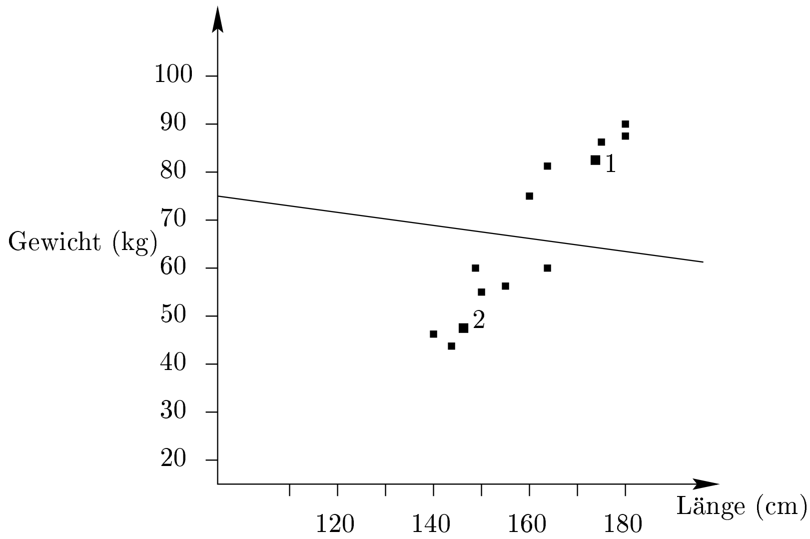


Abbildung 13: Ein-Bit Vektorquantisierer

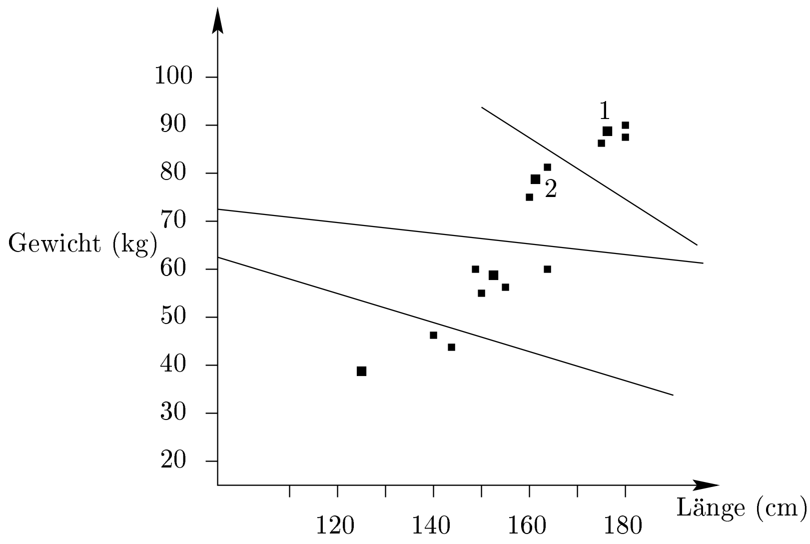


Abbildung 14: Zwei-Bit Vektorquantisierer

180,90	180,90	178,88(3)	178,88(3)	173,84(5)	173,84(5)
180,88	178,87(2)				
163,60	163,60	163,60	163,60	163,60	155,58(4)
110,21	110,21	110,21	110,21	110,21	110,21
148,60	153,59(2)	153,59(2)	150,57(3)	150,57(3)	
155,57					
160,75	163,78(2)	163,78(2)	163,78(2)		
150,55	150,55	150,55			
163,81					
140,46	143,45(2)	143,45(2)	143,45(2)	143,45(2)	143,45(2)
143,44					
175,86					

Abbildung 15: Der Ansatz von Equitz

Codebuchumfang K	$\log_2 K$	Bits pro Pixel	Kompressionsrate
16	4	0.25	32:1
64	6	0.375	21.33:1
256	8	0.50	16:1
1024	10	0.625	12.8:1

Tabelle 15: Verschiedene Kompresssionsmaße für ein Grauwertbild mit 8 Bit pro Pixel Auflösung

Codebuchumfang K	Overhead (Bits pro Pixel)
16	0.03125
64	0.125
256	0.50
1024	2.0

Tabelle 16: Overhead für die Codebuchübertragung

gen vom Mittelwert trainiert. Bei diesem "hybriden" Verfahren kann man natürlich auch einen Jayant-Quantisierer einsetzen.

Dem zweiten Problem kann durch Wahl geeigneter Datenstrukturen begegnet werden, z.B. einer baumartigen Strukturierung der Codevektoren.

Beide Probleme lassen sich natürlich mit "regelmäßigen" Codebüchern umgehen, z.B. dem gitterartigen aus Bild 5. Allerdings hat sich eine hexagonale (wabenartige) Struktur für zweidimensionale Codebücher als besser erwiesen.

8 Weitere verlustbehaftete Komprimierverfahren

Die Grundidee der

8.1 Differentialcodierung

ist es, die Korrelation aufeinanderfolgender Quellenausgaben dadurch auszunutzen, daß anstelle der Originalwerte die Differenzen zweier benachbarter Werte (codiert) übertragen werden, in der Hoffnung, daß dadurch der Wertebereich und die Varianz vermindert werden können.

Abb. 10.1 im Sayood enthält einmal die Kurve $f(t) = \cos(\frac{\pi}{6}t)$, $t \in [0,6]$ und auch noch die Differenzen $d_n = x_n - x_{n-1}$ mit $x_n = f(n/10)$, $n = 0, \dots 60$. Während die x_n zwischen -1 und 1 liegen, fallen die Differenzen d_n ins Intervall $[-0,2;0,2]$.[9] Benutzt man einen 2-Bit-Gleichquantisierer, so führt dies bei Übertragung der x_n zu einer Schrittweite von $2\dot{1}/2^2 = 0,5$[10], und der betreffende Quantisierungsfehler liegt im Intervall $[-0,25;0,25]$; dagegen nimmt man bei Übertragung der x_n die Schrittweite $0,1$, was den Quantisierungsfehler auf das Intervall $[-0,05;0,05]$ beschränkt.

Bei Bildern führt der Übergang zu Pixeldifferenzen oft zu Histogrammen, die der Laplace-Verteilung ähneln, s. Bild 10.2, 10.3 im Sayood. Eine Quantisierung des "Differenzenbildes" mit fünf oder gar vier Bit ist genauso gut wie die des Originalbilds mit sieben Bit.

Stimmt diese Bit-Analyse wirklich? Bislang haben wir das Phänomen der Fehlerfortpflanzung bei verlustbehafteter Übertragung vernachlässigt, was an folgendem Beispiel deutlich wird, wo wir einen Quantisierer mit sieben Ausgabewerten $\{-6,-4,-2,0,2,4,6\}$ benutzen; Tabelle 17 enthält die Ausgangsfolge, die Differenzenfolge, die sich aus der Differenzenfolge ergebene Quantisierungsfolge, die dadurch gelieferte Rekonstruktion der Originalwerte und schließlich die Abweichung von selbigen. Wie man sieht, scheint die Abweichung tendenziell immer größer zu werden.

Wir wollen diese Fehlerfortpflanzung jetzt formaler untersuchen. Dazu sei die Ausgangsfolge mit $\{x_n\}$, die Differenzenfolge mit $\{d_n\}$, die quantisierte Differenzenfolge mit $\{\hat{d}_n\}$ und die rekonstruierte Folge mit $\{\hat{x}_n\}$ bezeichnet. Startwert sei x_0, und es gelte $\hat{x}_0 = x_0$. Dabei gilt:

$$d_n = x_n - x_{n-1}, \text{ falls } n > 0 \tag{5}$$

$$\hat{d}_n = Q[d_n] = d_n + q_n, \ q_n \text{ ist der Quantisierfehler} \tag{6}$$

[9] Eine Ausnahme bildet der erste Wert $d_0 = x_0$.

[10] gemäß den Betrachtungen im Abschnitt 7.2

Folge $\{x_n\}$	6,2	9,7	13,2	5,9	8,0	7,4	4,2	1,8
Diff. $\{d_n\}$	6,2	3,5	3,5	-7,3	2,1	0,6	-3,2	-2,4
Quant. $\{\hat{d}_n\}$	6	4	4	-6	2	0	-4	-2
Rekonstr. $\{\hat{x}_n\}$	6	10	14	8	10	10	6	4
Abweichung	0,2	-0,3	-0,8	-2,1	-2	-2,6	-1,8	-2,2

Tabelle 17: Zur Fehlerfortpflanzung

$$\begin{aligned} \hat{x}_n &= \hat{x}_{n-1} + \hat{d}_n, \text{ falls } n > 0 & (7) \\ &= x_n + \sum_{k=1}^{n} q_k, & (8) \end{aligned}$$

denn $\hat{x}_n - \hat{x}_{n-1} = \hat{d}_n = x_n - x_{n-1} + q_n$ liefert mit $\hat{x}_0 = x_0$ per Teleskopsumme:

$$(\hat{x}_n - \hat{x}_{n-1}) + (\hat{x}_{n-1} - \hat{x}_{n-2}) + \ldots + (\hat{x}_1 - x_0) =$$
$$(x_n - x_{n-1}) + (x_{n-1} - x_{n-2}) + \ldots + (x_1 - x_0) + \sum_{k=1}^{n} q_k \,.$$

Eine leichte Modifikation verhindert diese sich im Term $\sum_{k=1}^{n} q_k$ ausdrückende Fehlerfortpflanzung: setze nun $d_n = x_n - \hat{x}_{n-1}$; damit gilt:

$$\hat{x}_n = \hat{x}_{n-1} + \hat{d}_n = \underbrace{\hat{x}_{n-1} + d_n}_{x_n} + q_n \,.$$

Hierbei kann $\hat{x}_{n-1}$ als ein Schätzer für x_n interpretiert werden. Diese Idee wird im folgenden verallgemeinert.

8.2 Prädikative Differentialcodierung

Bild 16 zeigt das Schema eines Codierers im prädikativen Differentialverfahren.[11] Q bezeichnet dabei den zur Codierung der Differenzenfolge verwendeten Quantisierer und P den "Prädikator" genannten Schätzer für x_n. Während am Ende des vorigen Abschnitts $p_n = \hat{x}_{n-1}$ galt, kann man im allgemeinen

$$p_n = f(\hat{x}_{n-1}, \hat{x}_{n-2}, \ldots, \hat{x}_0)$$

ansetzen. Bemerkenswert ist, daß der Codierer den Decodierer (im gestrichelt eingekästelten Bereich) simuliert.

Wie ist die Vorhersagefunktion f zu wählen? Da für die Differenzenfolge $d_n = x_n - p_n$ gilt, beeinflußt die Schätzfolge p_n offenbar u.a. die Varianz σ_d^2

[11] Dieses Schema wird englisch als "differential pulse code modulation", kurz DPCM, bezeichnet.

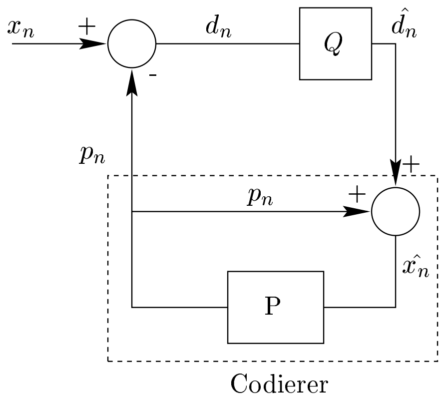


Abbildung 16: Der Codierer des prädikativen Differentialverfahrens

der Differenzfolge, die möglichst klein sein sollte, um eine gute (verlustbehaftete) Komprimierung durchführen zu können.

Unter den Annahmen

1. $p_n = f(\hat{x}_{n-1}, \hat{x}_{n-2}, \ldots, \hat{x}_0) \sim f(x_{n-1}, x_{n-2}, \ldots, x_0)$ und
2. $\{x_n\}$ ist stationär

kann man zeigen, daß σ_d^2 durch den bedingten Erwartungswert $E[x_n \mid x_{n-1}, \ldots, x_0]$ minimiert wird. Da dies immer noch zu kompliziert ist, nehmen wir weiter an, die Vorhersagefunktion sei linear, d.h.

$$p_n = \sum_{i=1}^{N} a_i \hat{x}_{n-i} \,;$$

N heißt auch Ordnung des Prädikators. Wegen der ersten Annahme ist also

$$\sigma_d^2 = E[(x_n - \sum_{i=1}^{N} a_i x_{n-i})^2]$$

zu minimieren. Hierzu setzen wir die partiellen Ableitungen bezüglich der a_i gleich Null:

$$\begin{aligned}
\frac{\partial \sigma_d^2}{\partial a_1} &= -2E\left[\left(x_n - \sum_{i=1}^{N} a_i x_{n-i}\right) x_{n-1}\right] = 0 \\
\frac{\partial \sigma_d^2}{\partial a_2} &= -2E\left[\left(x_n - \sum_{i=1}^{N} a_i x_{n-i}\right) x_{n-2}\right] = 0
\end{aligned}$$

$$\begin{aligned}
\vdots \quad & \quad \vdots \\
\frac{\partial \sigma_d^2}{\partial a_N} &= -2E\left[\left(x_n - \sum_{i=1}^{N} a_i x_{n-i}\right) x_{n-N}\right] = 0.
\end{aligned}$$

Unter Benutzung der Autokorrelationsfunktion

$$R_{xx}(k) = E[x_n x_{n+k}]$$

können wir so ansetzen:

$$\begin{aligned}
\sum_{i=1}^{N} a_i R_{xx}(i-1) &= R_{xx}(1) \\
\sum_{i=1}^{N} a_i R_{xx}(i-2) &= R_{xx}(2) \\
\vdots \quad & \quad \vdots \\
\sum_{i=1}^{N} a_i R_{xx}(i-N) &= R_{xx}(N)
\end{aligned}$$

Dies läßt sich auch in Matrixform schreiben:

$$\mathbf{RA} = \mathbf{P}$$

wobei

$$\mathbf{R} = \begin{bmatrix} R_{xx}(0) & R_{xx}(1) & R_{xx}(2) & \dots & R_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & R_{xx}(1) & \dots & R_{xx}(N-2) \\ R_{xx}(2) & R_{xx}(1) & R_{xx}(0) & \dots & R_{xx}(N-3) \\ \vdots & \vdots & \vdots & & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & R_{xx}(N-3) & \dots & R_{xx}(0) \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_N \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} R_{xx}(1) \\ R_{xx}(2) \\ R_{xx}(3) \\ \vdots \\ R_{xx}(N) \end{bmatrix}$$

Somit ergibt sich der Vektor der Prädikatorkoeffizienten durch:

$$\mathbf{A} = \mathbf{R}^{-1}\mathbf{P}$$

Natürlich können wir in der Praxis nicht annehmen, wir hätten bereits Kenntnis der genauen in Matrix **R** zusammgefaßten Autokorrelationswerte. Statt dessen verwendet man als Schätzer dieser Werte M Datenpunkte:

$$R_{xx}(k) = \frac{1}{M-k} \sum_{i=1}^{M-k} x_i x_{i+k}\,.$$

Quantisierer	Prädikatorordnung	SNR (dB)	SPER (dB)
2-Bit	keine	2.43	
	1	3.37	2.65
	2	8.35	5.9
	3	8.74	6.1
3-Bit	keine	3.65	
	1	3.87	2.74
	2	9.81	6.37
	3	10.16	6.71

Tabelle 18: Leistungswerte von DPCM-Systemen mit unterschiedlichen Quantisierern und Prädikatoren

Für eine Spracheingabe wurden auf diese Weise die Prädikatorkoeffizienten ermittelt. Diese sind natürlich von der Ordnung des Prädikators abhängig. Tabelle 18 enthält das Verhältnis SNR von Signal zur Verzerrung sowie das Verhältnis DPER von Signal zur Vorhersage, d.i.,

$$SPER(dB) = \frac{\sum_{i=1}^{M} x_i^2}{\sum_{i=1}^{M} (x_i - p_i)^2}$$

für dieses Sprachbeispiel, bei dem die Quelle (zu Vergleichszwecken) einmal mit einem 2-(bzw. 3-)Bit Quantisierer codiert wurde (daher keine Prädikatorordnung) und dann mit einem linearen Prädikator (der gemäß obigem Ansatz erhalten wurde) der Ordnung $N = 1, 2, 3$.

Gerade bei Sprache ist die Stationaritätsannahme sicher falsch, was es nahelegt, adaptiv zu quantisieren.

8.3 Adaptive Differentialcodierung

Vergegenwärtigt man sich nochmals Abb. 16, so erkennt man, daß es grundsätzlich zwei Komponenten gibt, die adaptiv sein können, nämlich der Quantisierer Q und der Prädikator P. Die weiter erkennbare Rückkopplung läßt eine Voradaptierung des Quantisierers nicht sinnvoll erscheinen. Die Rückadaptierung von Q kann mit einem Jayant-Quantisierer geschehen.

Beim Entwurf der Prädikatorkoeffizienten wurde stets von der Stationarität der Quelle ausgegangen, was (z.B. bei Sprache) i.allg. falsch ist. Daher erscheint eine Adaptierung der Prädikatorparameter sehr sinnvoll. Bei DPCM mit Voradaptierung des Prädikators (DPCM-APF) wird die Eingabe in Blöcke unterteilt, der Schätzer der Autokorrelationskoeffizienten je

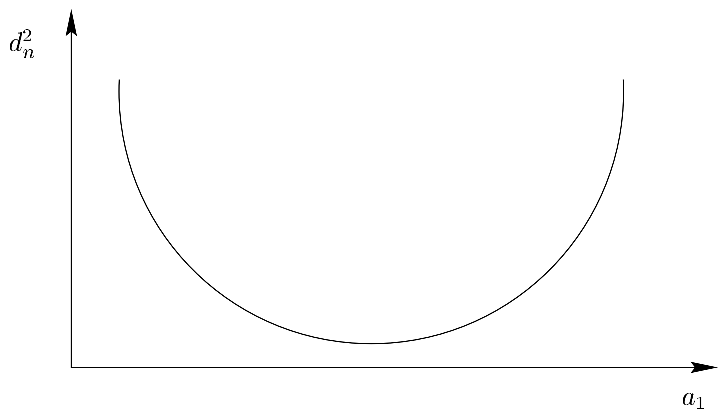


Abbildung 17: Fehlerfunktion in Abhängigkeit des Prädikatorparameters

Block berechnet und als Begleitinformation übertragen. Insbesondere bei Sprachübertragung kann DPCM-APF nicht tolerierbare Verzögerungen bewirken. Wir beschäftigen uns daher jetzt mit Rückadaptierung des Prädikators (DPCM-APB).

Der Fehler eines Prädikators erster Ordnung läßt sich durch

$$d_n^2 \quad = \quad (x_n - a_1 \hat{x}_{n-1})^2$$

messen. Fassen wir d_n^2 als Funktion von a_1 auf, so ergibt sich eine Parabel wir in Bild 17. Ist der Wert $a_1^{(n)}$ weit vom Optimum entfernt, so ist eine raschere Adaptierung erwünscht als wenn fast das Optimum erreicht ist. Diesen Effekt erreichen wir durch Anwendung der Formel

$$\begin{aligned}
a_1^{(n+1)} &= a_1^{(n)} - \alpha \frac{\partial d_n^2}{\partial a_1} \text{ mit} \\
\frac{\partial d_n^2}{\partial a_1} &= -2(x_n - a_1 \hat{x}_{n-1})\hat{x}_{n-1} \\
&= -2 d_n \hat{x}_{n-1}; \text{ also ist:} \\
a_1^{(n+1)} &= a_1^{(n)} + \alpha' d_n \hat{x}_{n-1}
\end{aligned}$$

für eine die Adaptivität beeinflussende positive Konstante α bzw. α'. Da die nichtquantisierten Werte d_n nur dem Codierer zur Verfügung stehen, ersetzen wir in jener Formel d_n durch $\hat{d}_n$:

$$a_1^{(n+1)} \quad = \quad a_1^{(n)} + \alpha \hat{d}_n \hat{x}_{n-1}$$

Für einen Codierer höherer Ordnung $N > 1$ erhält man ganz entsprechend durch Betrachtung von

$$d_n^2 = \left(x_n - \sum_{i=1}^{N} a_i \hat{x}_n - i\right)^2$$

sowie der partiellen Ableitungen dieser Funktion nach den a_j als Adaptionsregel für den j-ten Prädikatorkoeffizienten:

$$\begin{aligned}
a_j^{(n+1)} &= a_j^{(n)} + \alpha \hat{d}_n \hat{x}_{n-j} \text{ bzw. vektoriell} \\
\mathbf{A}^{(n+1)} &= \mathbf{A}^{(n)} + \alpha \hat{d}_n \hat{X}_{n-1} \text{ mit} \\
\hat{X}_n &= \begin{bmatrix} \hat{x}_n \\ \hat{x}_{n-1} \\ \vdots \\ \hat{x}_{n-N+1} \end{bmatrix}
\end{aligned}$$

8.4 Delta-Modulierung

Für die Sprachcodierung wird gerne der sog. Δ-Modulator verwendet, d.i. ein DPCM-Verfahren mit 1-Bit-Quantisierer. 1-Bit-Quantisierung bedeutet, daß der Codierer dem Decodierer quasi nur mitteilen darf, ob sich der aktuelle Eingabewert im Vergleich zum vorigen erhöht oder erniedrigt hat, und der Decodierer interpretiert diese Information, indem er $\Delta > 0$ zu seinem alten Wert hinzuaddiert bzw. davon subtrahiert. Um Fehler nicht allzu groß werden zu lassen, ist eine hohe Abtastrate erforderlich. Dennoch ist bei fixiertem 1-Bit-Quantisierer zu beobachten:

- Ist die Quelle fast konstant, so schwingt der Modulator.
- Bei steilem Signalanstieg oder -abfall kommt er nicht so schnell nach.

Diese nachteiligen Effekte lassen sich durch Wahl von Δ beeinflussen:

- Ist Δ klein, so beobachten wir schwaches Schwingen, aber schlechtes Nacheilen.
- Ist Δ groß, so sind auch die Schwingungen groß, aber das Nacheilverhalten ist gut.

Auch hier bietet sich eine adaptive Delta-Modulierung an; wir betrachten im folgenden die sog. CFDM (constant factor delta modulation). Δ_n bezeichne den vom Decodierer erhaltenen vorzeichenbehafteten Differenzterm im n-ten

Schritt. Da die nachstehend angegebene Adaption mit den Faktoren $M > 0$ nicht das Vorzeichen von Δ_n beeinflußt (dieses ist schließlich die eigentlich übertragene Information), ist folgende Festlegung nicht zirkulär:

$$\begin{aligned} s_n &= \begin{cases} 1 & \text{wenn } \Delta_n > 0 \\ -1 & \text{wenn } \Delta_n < 0 \end{cases} \\ \Delta_n &= \begin{cases} M\Delta_{n-1} & s_n = s_{n-1} \\ 1/M\Delta_{n-1} & s_n \neq s_{n-1} \end{cases} \end{aligned}$$

CFDM läßt sich verbessern, indem nicht nur die letzte Ausgabe des Codierers in Betracht gezogen wird. Der Leser mag sich klarmachen, was die folgenden Fälle "bedeuten":

$$\begin{aligned} s_n &\neq s_{n-1} = s_{n-2} \\ s_n &= s_{n-1} \neq s_{n-2} \\ s_n &= s_{n-1} = s_{n-2} \end{aligned}$$

Für die Sprachcodierung werden hierbei empfohlen:

$$\begin{array}{ccccccc} s_n & \neq & s_{n-1} & = & s_{n-2} & \Rightarrow & M = 0,4 \\ s_n & \neq & s_{n-1} & \neq & s_{n-2} & \Rightarrow & M = 0,9 \\ s_n & = & s_{n-1} & \neq & s_{n-2} & \Rightarrow & M = 1,5 \\ s_n & = & s_{n-1} & = & s_{n-2} & \Rightarrow & M = 2,0 \end{array}$$

8.5 Teilbandcodierung

8.5.1 Einführung

Beobachten wir einmal folgende Beispielfolge $\{x_n\}$:

$$10 \quad 14 \quad 10 \quad 12 \quad 14 \quad 8 \quad 14 \quad 12 \quad 10 \quad 8 \quad 10 \quad 12$$

Bei Anwenden eines DPCM-Schemas müßte man die folgende Differenzenfolge codieren:

$$10 \quad 4 \quad -4 \quad 2 \quad 2 \quad -6 \quad 6 \quad -2 \quad -2 \quad -2 \quad 2 \quad 2$$

Abgesehen vom ersten Wert liegen alle Differenzen zwischen -6 und $+6$. Ein m-Bit Gleichquantisierer würde eine Schrittweite $\Delta = 12/2^m$ benutzen, was zu einem maximalen Quantisierfehler von $\Delta/2 = 6/2^m$ führte. Beachtet man, daß anstelle dieser Differenzen ohne Informationsverlust auch $z_n = \frac{x_n - x_{n-1}}{2}$ übertragen werden kann, so halbiert dies den Quantisierfehler.

Betrachten wir hingegen die Folge $y_n = \frac{x_n + x_{n-1}}{2}$ der Mittelwerte, so ergibt sich als Differenzenfolge der y_n:

$$10 \quad 2 \quad 0 \quad -1 \quad 2 \quad -2 \quad 0 \quad 2 \quad -2 \quad -2 \quad 0 \quad 2$$

Da diese Differenzen lediglich zwischen -2 und $+2$ liegen, ergibt sich ein maximaler Quantisierfehler für einen m-Bit Gleichquantisierer von $2/2^m$. Wie auch intuitiv zu erwarten, ist die Mittelwertfolge "glatter" als die Originalwertfolge.

Überträgt man die Codierungen der sich offenbar als angenehmer erweisenden Folgen y_n und z_n, so läßt sich hieraus durch Addition die eigentlich zu übertragende Folge x_n rekonstruieren. Beachtet man überdies, daß sich alle x_n-Werte aufgrund der Zusammenhänge

$$\begin{aligned} x_{2n-1} &= y_{2n} - z_{2n} \text{ und} \\ x_{2n} &= y_{2n} + z_{2n} \end{aligned}$$

wiedergewinnen lassen lediglich durch Übertragen der "halben" Folgen y_{2n} und z_{2n}, und auch die hierfür zu betrachtende Differenzenfolge $y_{2n} - y_{2n-2}$ verhält sich angenehm.

Die Idee, eine gegebene Folge durch sog. *Filter* in *Bänder* genannte Bestandteile zu zerlegen, die dann (evtl. mit jeweils angepaßten unterschiedlichen Verfahren codiert) getrennt übertragen werden, ist Grundgedanke der Teilbandcodierung. Im Beispiel enthielt das Band der Mittelwertsfolge "langfristige Informationen", also niederfrequente Anteile, und das Band der Differenzenfolge hochfrequente Anteile. Niederfrequenzbänder lassen sich oft gut mit DPCM-Technik komprimieren.

8.5.2 Frequenzfilter

In Verallgemeinerung des obigen Beispiels wollen wir jetzt Filter betrachten, die eine Eingabefolge $\{x_n\}$ in eine Ausgabefolge $\{y_n\}$ vermöge

$$y_n = \sum_{i=0}^{N-1} a_i x_{n-i} + \sum_{i=1}^{M} b_i y_{n-i}$$

überführen. Auf die spezielle Impuls-Folge

$$x_n = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

"antwortet" ein Filter mit der "Impulsantwort" $\{h_n\}$. Sind alle b_i Null, so handelt es sich um einen Filter mit endlicher Impulsantwort (FIR: finite impulse response) mit höchstens N Nicht-Null-Werten $h_0, \ldots, h_{N-1}$, andernfalls um einen mit unendlicher Impulsantwort (IIR: infinite impulse response).[12]

[12] Wir beschäftigen uns hier nur mit eindimensionalen Signalen und führen den zweidimensionalen Fall hierauf zurück. Der allgemeine mehrdimensionale Fall ist z.B. in [6] dargestellt.

Beispielsweise errechnet man als Impulsantwort für den durch die Nicht-Null-Werte $a_0 = 1,25$ und $a_1 = 0,5$ spezifierten Filter:

$$\begin{aligned} h_0 &= a_0 x_0 + a_1 x_{-1} = 1,25 \\ h_1 &= a_0 x_1 + a_1 x_0 = 0,5 \\ h_n &= 0 \text{ sonst} \end{aligned}$$

Für den durch die Nicht-Null-Werte $a_0 = 1$ und $b_1 = 0,9$ spezifierten Filter bekommt man als Impulsantwort:

$$\begin{aligned} h_0 &= a_0 x_0 + b_1 h_{-1} = 1(1) + 0,9(0) = 1 \\ h_1 &= a_0 x_1 + b_1 h_0 = 1(0) + 0,9(1) = 0,9 \\ h_2 &= a_0 x_2 + b_1 h_1 = 1(0) + 0,9(0,9) = 0,81 \\ &\vdots \\ h_n &= (0.9)^n \end{aligned}$$

Ähnlich erhält man als Impulsantwort h_n bzw. h'_n des "Mittelwertfilters y_n" bzw. des "Differenzfilters z_n" aus dem Beispiel im vorigen Abschnitt:

$$h_n = \begin{cases} 0,5 & n = 0 \\ 0,5 & n = 1 \\ 0 & \text{sonst} \end{cases} \quad h'_n = \begin{cases} 0,5 & n = 0 \\ -0,5 & n = 1 \\ 0 & \text{sonst} \end{cases}$$

Allgemein legt die Impulsantwort die Filterfunktion in folgender Weise fest:

$$y_n = \sum_{k \geq 0} h_k x_{n-k}$$

Idealisiert kann man sich ein Teilbandcodierverfahren wie in Bild 18 vorstellen: Auf der Codiererseite wird der einkommende Signalstrom zunächst durch eine Filterbank in M (sich evtl. überlappende) Teilbänder zerlegt; jeder gefilterte Signalstrom wird dann "dezimiert", d.h., nur jedes M-te Datum wird dem eigentlichen Codierer weitergeleitet[13]. Um unterschiedliche Charakteristika verschiedener Bänder desselben Signals auszunutzen, werden die für die Einzelbänder benutzten Codierverfahren unterschiedlich sein; typischerweise sind niederfrequente Bänder DPCM-quantisiert und höherfrequente skalar- oder vektorquantisiert. Im Decodierer müssen umgekehrt die rekonstruierten Signale der verschiedenen Bänder wieder zusammengesetzt werden. Die Synthesefilter sind natürlich im allgemeinen von den Analysefiltern verschieden.

[13] In dem Beispiel des vorigen Abschnitts hatten wir gesehen, das dies für die Rekonstruktion hinreichend ist, und jene Überlegung läßt sich verallgemeinern. Anstelle von "Dezimation" spricht man auch von "Unterabtastung".

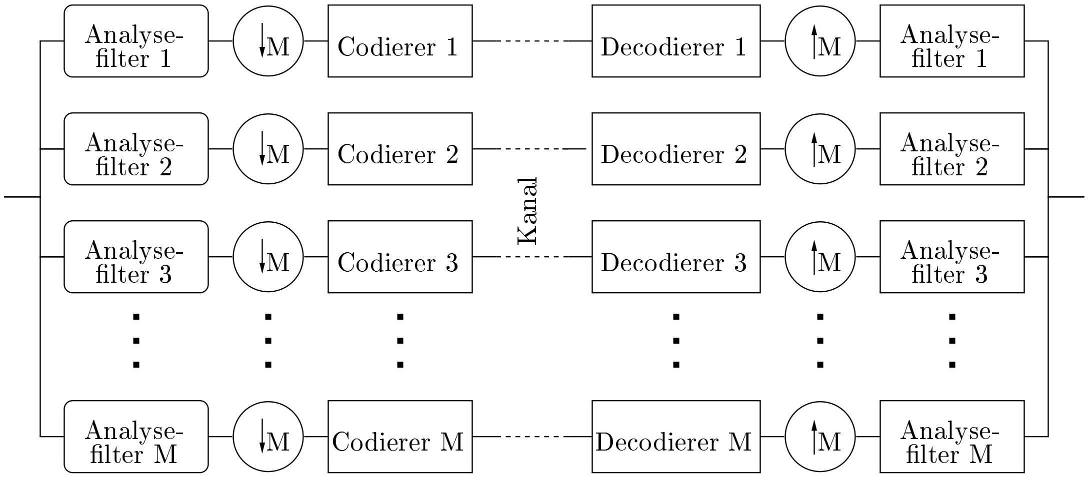


Abbildung 18: Blockdiagramm zur Teilbandcodierung

10	14	10	12	14	8	14	12
10	12	8	12	10	6	10	12
12	10	8	6	8	10	12	14
8	6	4	6	4	6	8	10
14	12	10	8	6	4	6	8
12	8	12	10	6	6	6	6
12	10	6	6	6	6	6	6
6	6	6	6	6	6	6	6

Tabelle 19: Ein Beispiel“bild”

Dezimierte Tiefpaßausgabe				Dezimierte Hochpaßausgabe			
5	12	13	11	5	-2	1	3
5	10	11	8	5	-2	1	2
6	9	7	11	6	-1	1	1
4	5	5	7	4	-1	-1	1
7	11	7	5	7	-1	-1	1
6	10	8	6	6	-2	-2	0
6	8	6	6	6	-2	0	0
3	6	6	6	3	0	0	0

Tabelle 20: Ein Beispielbild — gefilterte und dezimierte Zeilen

Low-Low Image				Low-High Image			
2.5	6	6.5	5.5	2.5	6	6.5	5.5
5.5	9.5	9	9.5	0.5	-0.5	-2	1.5
5.5	8	6	6	1.5	3	1	-1
6	9	7	6	0	1	-1	0
High-Low Image				**High-High Image**			
2.5	-1	0.5	1.5	2.5	-1	0.5	1.5
5.5	-1.5	1	1.5	0.5	0.5	0	-0.5
5.5	-1	-1	1	1.5	0	0	0
6	0	0	0	0	0	1	0

Tabelle 21: Ein Beispielbild — 4 gefilterte und dezimierte Teilbilder

Die Tab. 19 bis 21 zeigen die Arbeitsweise des "Tiefpaßfilters" (also des Mittelwertfilters) und des "Hochpaßfilters" (des Differenzenfilters) aus dem Beispiel im vorigen Abschnitt, so wie diese üblicherweise im Falle von Bildern eingesetzt werden: zunächst wird zeilenweise, dann spaltenweise gefiltert. Leider sind in diesem Mini-Beispiel die Entstellungen an den Rändern noch überdeutlich. Davon einmal abgesehen, soll auch hier schon klar werden, daß die Hauptinformation des Bildes im niedrigfrequenten (oben links dargestellten) Teilbild vorhanden ist und "fast keine" Information im hochfrequenten Teilbild (unten rechts) in Tab. 21. "Gute" Filter konzentrieren möglichst viel Energie im niedrigfrequenten Teilbild.

8.5.3 Filterbänke

Filterbänke sind oft kaskadenartig aus Paaren von Tief- und Hochpaßfiltern (man spricht auch von Glättungs- und Differenzierfiltern) aufgebaut. Besonders beliebt sind Spiegelfilter (quadrature mirror filters QMF); ist $\{h_n\}$ die Impulsantwort des Tiefpaßfilters, so ist $\{(-1)^n h_{N-1-n}\}$ die Impulsantwort des Hochpaßfilters. Wie oben nachgerechnet, sind die Filter aus dem einführenden Beispiel im vorigen Abschnitt von dieser Art. Die beliebten Filter von Johnston und Smith-Barnwell sind ebenfalls von dieser Gestalt, so daß zu ihrer Angabe die Impulsantwort des Tiefpaßfilters genügt.

Natürlich beeinflußt die Wahl der Filterbank auch die Art und Weise der Quantisierung der einzelnen Teilbänder. Nehmen wir an, R Bits sollen pro Datum höchstens übertragen werden, und die Eingabesignalvarianz des k-ten Quantisierers ($1 \leq k \leq M$) sei $\sigma_{y_k}^2$, so arbeitet folgendes Verfahren gut für die Verteilung der zur Verfügung stehenden R Bits auf die Teilbänder:

1. Berechne $S_k = \sigma_{y_k}^2$ und setze $R_k = 0$ für $1 \leq k \leq M$.

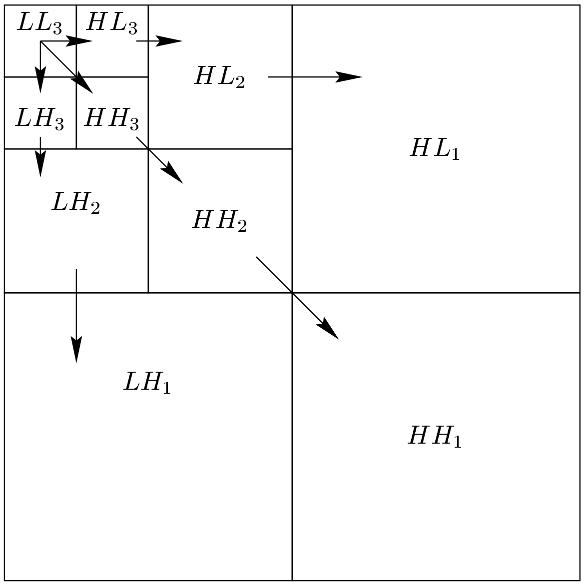


Abbildung 19: Abkömmlingsbaum à la Shapiro

2. Wähle ein k', so daß $S_{k'}$ maximal für $1 \leq k' \leq M$.
3. Inkrementiere $R_{k'}$; dividiere $S_{k'}$ durch zwei.
4. Dekrementiere R. Falls $R > 0$, gehe zu Schritt 2.

Nach Durchlauf des Verfahrens sollten R_k Bits für das k-te Band genommen werden. Als Beispiel wären jetzt die Bilder 11.12, 11.14, 11.15 und 11.16 aus dem Sahood nett!

8.5.4 Shapiros EZW-Algorithmus

Eine andere, adaptive Idee zur Bitverteilung bei der Bildkompression mit Teilbandverfahren ist im EZW-Algorithmus[14] von Shapiro enthalten [11]. Wie oben erläutert, wird ein Bild zeilen- und spaltenweise mit einem Tief- bzw. Hochpaßfilter in vier Bandbereiche zerlegt; für den Low-Low-Bereich, in dem sich ja die Energie bei geeigneter Filterwahl konzentrieren sollte, kann man nun diese Zerlegung wiederholen, was bei dreimaliger Wiederholung zu der Aufteilung des "gefilterten und dezimierten" Bildes führt, die in Abb. 20 wiedergegeben ist.

[14]EZW steht für "Embedded Zerotree Wavelet"; im Ggs. zum Namen ist das Grundprinzip des Vorgehens nicht auf Wavelets beschränkt.

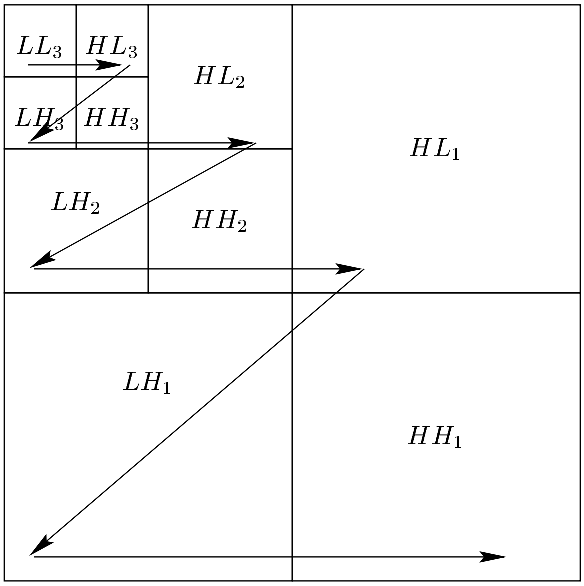


Abbildung 20: Abtasten von Filterkoeffizienten à la Shapiro

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Abbildung 21: Ein dreimal wiederholt gefiltertes und dezimiertes “Bild”

Wir erläutern den Algorithmus am besten anhand eines Beispiels. Es seien die in Abb. 21 gegebenen gefilterten und dezimierten Werte vorgegeben. Der betragsmäßig größte Wert ist 63. Als initialen Schwellwert T_0 wählen wir daher dessen gerundete Hälfte, d.i. $T_0 = 32$. Im ersten Hauptdurchgang wird nun das Bild, wie in 20 durch den Pfeil angegeben, abgetastet. Für Teilbilder wird rekursiv dieselbe Abtastreihenfolge gewählt. In den Hauptdurchgängen wird ein Strom von Symbolen aus { POS, NEG, IZ, ZTR } erzeugt; diese Zeichen sind natürlich mit je zwei Bit codierbar. Dabei bedeutet POS, daß der abgetastete Wert größer als der aktuelle Schwellwert T_k ist, NEG, daß er negativ und kleiner als $-T_k$ ist und IZ bzw. ZTR, daß er im Intervall $[-T_k, T_k]$ liegt. ZTR (zerotree) bedeutet überdies, daß alle "Abkömmlinge" im Sinne des in Bild 19 mit Pfeilen dargestellten Baumes (der im übrigen wieder rekursiv fortgesetzt zu denken ist, s.u. Beispiel) im Intervall $[-T_k, T_k]$ liegen und so einen "Nullbaum" bilden; dahingegen bezeichnet IZ eine isolierte Null (isolated zero).

Der erste Hauptdurchgang ist in Tabelle 22 aufgelistet. Als ersten Wert erhalten wir so 63. Dieser ist größer als der Schwellwert 32, weshalb Symbol POS generiert wird. Da -34 kleiner als -32 ist, wird als nächstes NEG erzeugt. -31 ist im Intervall $[-32, 32]$, im darunterhängenden Teilbaum liegt aber noch ein "signifikanter Wert", nämlich 47, so daß eine isolierte Null vorliegt und IZ erzeugt wird. Dahingegen ist sowohl der Eintrag im HH_3-Band wie alle seine Abkömmlinge (in den Bändern HH_2 und HH_1) insignifikant, weshalb wir bei (3) ein ZTR ausgeben. Insbesondere bedeutet dies, daß wir weiteren Symbole mehr für Einträge aus dem HH_2- und HH_1-Band erzeugen werden in diesem Hauptdurchgang.

Jetzt wird das Teilband HL_2 abgetastet. Bei (4) ist zu bemerken, daß die Abkömmlinge des Eintrags 10 eben $\{-12, 7, 6, 1\}$ sind, was durch rekursive Forsetzung des in Bild 19 dargestellten Abkömmlingsbaums zu sehen ist, und diese sämtlich dem Betrage nach unter dem Schwellwert liegen, weshalb ein Nullbaum diagnostiziert wird. Bei (5) (innerhalb der Analyse des LH_2-Bands) merken wir an, daß zwar 14 selbst unterhalb des Schwellwerts liegt, das Abkömmlingsteilband $\{-1, 47, -3, 2\}$ jedoch mit 47 einen signifikanten Wert enthält, weshalb IZ auszugeben ist. Beachte bei (6), daß das HH_2-Band bereits berücksichtigt wurde und daher gleich zur Codierung des HL_1-Bands geschritten wird. Da die HL_1 und LH_1-Bänder keine Abkömmlinge besitzen, können wir die ZTR und IZ-Symbole zu einem einzigen Z-Symbol verschmelzen und so noch kompakter codieren.

Tabelle 23 enthält die Ergebnisse der sich anschließenden ersten Verfeinerung, bei der für die vier signifikanten Werte (aus dem ersten Hauptdurchgang) angegeben wird, ob sie (dem Betrage nach) im Intervall $[32, 48)$ oder in $[48, 64)$ liegen.

Kommentar	Teilband	Wert	Symbol	rekonstruierter Wert
(1)	LL_3	63	POS	48
	HL_3	-34	NEG	-48
(2)	LH_3	-31	IZ	0
(3)	HH_3	23	ZTR	0
	HL_2	49	POS	48
(4)	HL_2	10	ZTR	0
	HL_2	14	ZTR	0
	HL_2	-13	ZTR	0
	LH_2	15	ZTR	0
(5)	LH_2	14	IZ	0
	LH_2	-9	ZTR	0
	LH_2	-7	ZTR	0
(6)	HL_1	7	Z	0
	HL_1	13	Z	0
	HL_1	3	Z	0
	HL_1	4	Z	0
	LH_1	-1	Z	0
(7)	LH_1	47	POS	48
	LH_1	-3	Z	0
	LH_1	-2	Z	0

Tabelle 22: Ergebnisse des ersten Hauptdurchgangs

Koeffizienten-größe	Symbol	Rekonstruktions-größe
63	1	56
34	0	40
49	1	56
47	0	40

Tabelle 23: Die erste Verfeinerung

Beim nächsten Hauptdurchgang werden nur die bislang für insignifikant gehaltenen Werte berücksichtigt; die bislang signifikanten Werte werden hingegen in diesem Durchgang als Nullen (im Sinne der Nullbaumkonstruktion) gehalten. Als Schwellwert nehmen wir $T_1 = T_0/2 = 16$. Im zweiten Verfeinerungsdurchgang hingegen werden alle aus den bisherigen (beiden) Hauptdurchgängen gewonnenen Werte verfeinert, was also zu einer Intervalleinteilung $[16, 24)$, $[24, 32)$, $[32, 40)$, $[40, 48)$, $[48, 56)$ und $[56, 64)$ führt.

Insbesondere wird so hoffentlich die Bedeutung des Wortes "embedded" deutlich: Jeder hochauflösendere Code enthält sämtliche niederauflösenden Codes eines Bildes als Präfix; m.a.W., EZW eignet sich sehr gut zur fortschreitenden Bildübertragung.

8.5.5 Filter aus Transformationen

Wie kann man *systematisch* ein Eingabesignal in Frequenzbereiche zerlegen? Eine Möglichkeit ist ziemlich bekannt: die Fourier-Entwicklung bzw. -Transformation. Jede reelle (oder komplexe) Funktion f mit Periode T läßt sich darstellen in der Form:

$$\begin{aligned}
f(t) &= \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(n\frac{2\pi}{T}t\right) + \sum_{n=1}^{\infty} b_n \sin\left(n\frac{2\pi}{T}t\right) \text{ mit} \\
a_n &= \frac{1}{T}\int_0^T f(t)\cos\left(n\frac{2\pi}{T}t\right)dt, \\
b_n &= \frac{1}{T}\int_0^T f(t)\sin\left(n\frac{2\pi}{T}t\right)dt; \text{alternativ:} \\
f(t) &= \sum_{-\infty}^{\infty} c_n e^{jn\frac{2\pi}{T}t} \text{ mit} \\
c_n &= \frac{1}{T}\int_0^T f(t)e^{-jn\frac{2\pi}{T}t}dt
\end{aligned}$$

So erhält man nicht nur eine "Sinusentwicklung" von periodischen Funktionen, sondern auch eine "Sinus-Approximation", indem man die Reihenentwicklung nach endlich vielen Gliedern abbricht[15] Diese ist wichtig, da —elektrophysikalisch bedingt—, "physikalische Filter" keine "reinen" Bandpaßfilter sind, sondern diese immer nur durch Schwingungsfunktionen annähern können.

Eine lediglich auf einem Intervall $[0, T]$ definierte Funktion läßt sich leicht "periodisch fortsetzen".[16] Da wir zumeist mit "diskreten Funktionen", sprich

[15] Für quadratisch integrierbare Funktionen ist Konvergenz garantiert.

[16] Betrachtet man den Fall "$T \to \infty$", so erhält man die Fouriertransformierte von f.

Folgen $\{x_n\}$ der Länge N beschäftigen, ist für uns die diskrete Fourier-Transformation wichtig:

$$\begin{aligned} c_k &= \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{\frac{-j2\pi kn}{N}} \\ x_n &= \sum_{k=1}^{N-1} c_k e^{\frac{j2\pi kn}{N}} \end{aligned}$$

Im folgenden Kapitel werden wir hierauf zurückkommen.

Ein flexibleres und modernes Instrument zur Signalzerlegung liefern *Wavelets*. Bei ihnen steht sowohl der Zeit- als auch der Frequenzbereich parametrisiert zur Verfügung. Sehr beliebt sind die sog. Haar-Wavelets. Aus der einfachen "Urfunktion" (mother wavelet)

$$\psi_{0,0}(x) = \begin{cases} 1 & 0 \le x < \frac{1}{2} \\ -1 & \frac{1}{2} \le x < 1 \\ 0 & \text{sonst} \end{cases} \quad \text{wird durch}$$

$$\begin{aligned} \psi_{j,k}(x) &= \psi_{0,0}(2^j x - k) \\ &= \begin{cases} 1 & k2^{-j} \le x < (k+\frac{1}{2})2^{-j} \\ -1 & (k+\frac{1}{2})2^{-j} \le x < (k+1)2^{-j} \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

eine Schar von Funktionen. j kann als Streckparameter (Frequenzparameter) und k als Verschiebeparameter (Zeitparameter) gedeutet werden.

Um eine Idee von der Arbeitsweise von Wavelets zu erhalten, betrachten wir exemplarisch nun als Urfunktion

$$\phi_{0,0}(x) = \begin{cases} 1 & 0 \le x < 1 \\ 0 & \text{sonst} \end{cases}$$

sowie die erzeugte Schar

$$\phi_{j,k}(x) = \phi_{0,0}(2^j x - k)\,.$$

Angenommen, wir wollten eine Funktion $f : [0, N] \to \mathbb{R}_+$ approximieren. In erster Näherung setzen wir

$$\begin{aligned} \phi_f^0(t) &= \sum_{k=0}^{N-1} c_{0,k}\phi_{0,k} \quad \text{mit} \\ c_{0,k} &= \int_k^{k+1} f(t)[\phi_{0,k}(t)]dt \end{aligned}$$

Durch Wahl eines höheren Frequenzparameters können wir die Genauigkeit der Approximation steigern:

$$\begin{aligned} \phi_f^j(t) &= \sum_{k=0}^{2^j N-1} c_{j,k}\phi_{0,k} \quad \text{mit} \\ c_{j,k} &= 2^j \int_{k/2^j}^{(k+1)/2^j} f(t)dt\,. \end{aligned}$$

Offensichtlich gilt

$$c_{j-1,k} = 1/2(c_{j,2k} + c_{j,2k+1})\,. \tag{9}$$

Unser Ziel ist es, Funktionen in Bestandteile (mit evtl. unterschiedlichen Charakteristika) zu zerlegen. Wenn nun ϕ_f^1 die Funktion f genügend approximiert, kann man ϕ_f^0 als niederfrequenten Anteil betrachten und muß dann die Differenz $\phi_f^1 - \phi_f^0$ diskutieren. Es gilt wegen Gleichung (9):

$$\phi_f^1(t) - \phi_f^0(t) = \begin{cases} c_{0,k} - c_{1,2k} = -1/2c_{1,2k} + 1/2c_{1,2k+1} & k \le t < k+1/2 \\ c_{0,k} - c_{1,2k+1} = 1/2c_{1,2k} - 1/2c_{1,2k+1} & k+1/2 \le t < k+1 \end{cases}$$

M.a.W., die Differenz läßt sich leicht mit Haar-Wavelets ausdrücken, nämlich:

$$\phi_f^1(t) - \phi_f^0(t) = \underbrace{(-c_{1,2k} + c_{1,2k+1})}_{b_{0,k}}\psi_{0,k}(t)\,.$$

Die $2N$-Punkte Folge $c_{1,k}$ kann so in zwei N-Punkt Folgen $c_{0,k}$ und $b_{0,k}$ zerlegt werden; die zweite Folge kann als Faktoren von Wavelets interpretiert werden. Allgemein läßt sich jede Funktionenschar $\phi_{j,k}$, die gewissen Skalier-, Darstellungs- und Integrabilitätsbedingungen genügt, dafür benutzen, eine zugehörige Wavelet-Familie zu definieren.

Aus der Darstellungsbeziehung

$$\phi_{0,0}(t) = \sum h_n \phi_{1,n}(t)$$

gewinnt man die Impulsantwort h_n des Glättungsfilters, denn $\phi_{0,0}$ kann als stetige Form des Impulses gesehen werden, und die Darstellung

$$\psi_{0,0}(t) = \sum (-1)^{N-n-1} h_n \phi_{1,n}(t)$$

des zughörigen Ur-Wavelets liefert die Impulsantwort des Differenzierfilters. Beliebt sind insbesondere die so erhaltenen Daubechies- und Coiflet-Filter. Wavelets liefern Spiegelfilter.

Wichtig ist schließlich, daß man die Aufspaltung des Ursignals in Glätte- und Differenzanteil weitertreiben kann, indem der Glätteanteil rekursiv weiter aufgespalten wird. Die Differenzanteile m-ter Stufe lassen sich dann durch die Wavelets $\psi_{m,k}$ darstellen. Die so mögliche rekursive "Multiresolutionsanalyse" ist einer der wesentlichen Vorteile Wavelet-basierter Zeit/Frequenz-Analyse gegenüber dem (älteren) Fourier-Ansatz. Diese Zusammenhänge sind auch sehr schön abstrakt im siebenten und achten Kapitel von [4] dargestellt. Populr̈wissenschaftliche Artikel sind [7, 14].

8.6 Fraktale Codierung

Der mathematische Grundgedanke dieser Codierungstechnik ist sehr einfach: Statt ein "Datum" zu übertragen, übertrage man die Parameter einer kontrahierenden Abbildung f, deren Fixpunkt eben dieses Datum ist. Der Decodierer kann nun durch ein einfaches Iterationsverfahren (berechne $y_i = f(y_{i-1})$ ausgehend von einem beliebigen "Punkt" y_0 solange, bis eine gewünschte Genauigkeit erreicht wird) das zu übertragende Datum rekonstruieren. Nach dem Banachschen Fixpunktsatz funktioniert diese Idee prinzipiell für jeden vollständigen metrischen Raum, zum Beispiel auf dem $\mathbb{R}^2$ mit dem Euklidischen Abstand als Metrik ρ_E. Iterierte Funktionensysteme (IFS) gestatten die Beschreibung von Bildern als Fix"punkte".

Es sei (X, ρ) ein metrischer Raum. Die *r-Umgebung* von $A \subseteq X$ ist gegeben durch:

$$B_r(A) := \{y \in X \,|\, (\exists x \in A)(\rho(x,y) < r)\} (= \bigcup_{x \in A} B_r(x)).$$

Bezeichne $\mathcal{K}(X, \rho)$ die Menge aller nicht leeren aber kompakten Teilmengen von (X, ρ). Es seien $A, B \in \mathcal{K}(X, \rho)$. Die Abbildung

$$\rho_H(A, B) := \inf\{r : A \subseteq B_r(B) \text{ und } B \subseteq B_r(A)\}$$

ist eine Metrik auf $(\mathcal{K}(X, \rho), \rho_H)$. Sie heißt *Hausdorff-Metrik*. Ein *IFS F* (auf X), ist gegeben durch eine endliche Liste $F = (F(1), \ldots, F(n))$ von kontrahierenden Ähnlichkeitsabbildungen. Zugeordnet ist die *Liste der Ähnlichkeitsfaktoren* $(r_{F(1)}, \ldots, r_{F(n)})$. Jedes IFS $F = (F(1), \ldots, F(n))$ auf (X, ρ) läßt sich als Abbildung $\mathcal{K}(X, \rho) \to \mathcal{K}(X, \rho)$ auffassen durch

$$F(A) := \bigcup_{i=1}^{n} F(i)(A).$$

F ist in dieser Auffassung eine Kontraktion auf $(\mathcal{K}(X, \rho), \rho_H)$ mit einem Kontraktionsfaktor $r_F := \max_{i=1}^{n} r_{F(i)} < 1$ ist. Nach dem Banachschen Fixpunktsatz bestimmt F so eindeutig einen Fix"punkt" $A_F \in \mathcal{K}(X, \rho)$, falls (X, ρ) (und damit $(\mathcal{K}(X, \rho), \rho_H)$) vollständig ist.

Da sich der Abstand des Fixpunktes x_f einer kontrahierenden Abbildung f (mit Kontraktionsfaktor r_f) auf einem vollständigen metrischen Raum (X, ρ) von einem beliebigen Punkt $x \in X$ abschätzen läßt ($\rho(x, x_f) \leq \frac{1}{1-r_f} \cdot \rho(x, f(x))$), ist die n-te Iterierte $f^n(x)$ höchstens $\frac{r_f^n}{1-r_f} \cdot \rho(x, f(x))$ vom Fixpunkt entfernt, was[17] eine recht einfache Abschätzung des ansonsten nicht so einfach zu bestimmenden Hausdorff-Abstandes liefert und überdies ein schnelles Abbruchkriterium bei der Decodierung "fraktalen Codes" liefert.

Als Beispiel betrachten wir die Koch-Kurve, als Raum nehmen wir $X = [0,1] \times [0,1]$ mit der Euklidischen Metrik und als IFS $F = (F(1), F(2), F(3), F(4))$, wobei die einzelnen Abbildungen wie folgt definiert sind.

$$\begin{aligned}
F(1)(x_1, x_2) &= \frac{1}{3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \\
F(2)(x_1, x_2) &= \frac{1}{3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \frac{2}{3} \\ 0 \end{pmatrix}, \\
F(3)(x_1, x_2) &= \frac{1}{3} \begin{pmatrix} \cos(60^\circ) & -\sin(60^\circ) \\ \sin(60^\circ) & \cos(60^\circ) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \frac{1}{3} \\ 0 \end{pmatrix}, \\
F(4)(x_1, x_2) &= \frac{1}{3} \begin{pmatrix} \cos(-60^\circ) & -\sin(-60^\circ) \\ \sin(-60^\circ) & \cos(-60^\circ) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{3}\sin(60^\circ) \end{pmatrix}.
\end{aligned}$$

Der Deutlichkeit halber wollen wir in Abb. 22 mit der kompakten Menge $\{(x, 0) | 0 \leq x \leq 1\}$ starten. Beachte, daß der Hausdorff-Abstand der fünften Iteration von der Koch-Kurve sich mit $\frac{1}{3^5(1-1/3)} \frac{1}{3} \approx 0{,}002$ abschätzen läßt.

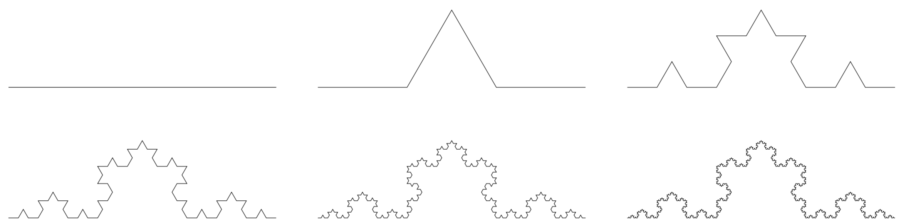


Abbildung 22: Die Koch-Kurve ist überall stetig, nirgends differenzierbar.

Wie auch anhand der Koch-Kurve deutlich wird, haben IFS-Fraktale stets eine "selbstähnliche" Struktur, die zwar ansatzweise –wie B. Mandelbrot herausgearbeitet hat– sich auch "in der Natur" und damit auch bei natürlichen Bildern anzutreffen ist, sicher aber nicht in der durch IFS bedingten "Reinkultur". Andererseits wird die wesentliche Idee eines auf "Selbstähnlichkeit" abhebenden Kompressionsverfahrens klar: Die Übertragung der viermal je

[17] In der Literatur über Fraktale heißt dieser Zusammenhang auch "collage theorem".

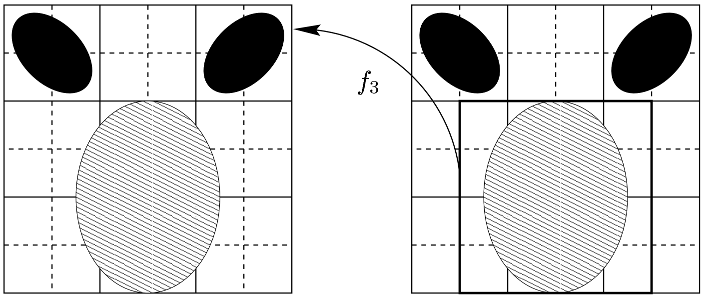


Abbildung 23: Bereichsblöcke und Urbildblöcke

6 Zahlen, die jeweils die affinen Abbildungen der Koch-Kurve spezifieren, ist wesentlich effizienter als die Übertragung beispielsweise des PS-Files der letzten Approximation in Bild 22. Des weiteren ergibt sich das Problem, wie man denn zu einem gegebenen Bild die Parameter einer es spezifizierenden Schar von affinen Abbildungen generieren kann.

Eine erste praktikable Lösung dieser Probleme stammt von A. Jacquin. Hierbei wird zunächst das Bild in sog. Bereichsblöcke R_k unterteilt. Für jeden Bereichsblock wird im Bild selbst ein Urbildblock D_k (und eine vermittelnde kontrahierende Abbildung f_k mit $R_k = f_k(D_k)$) gesucht. Die Urbildblöcke erhält man, indem man ein $K \times K$-Fenster mit Schrittweite $K/2$ oder $K/4$ über das Bild gleiten läßt und das "günstigste" Teilbild D_k auswählt, s. Bild 23. Da nicht immer die Existenz eines optimalen Urbilds gewährleistet ist, handelt man sich hierbei einen systematischen Fehler ein, der dieses Verfahren verlustbehaftet macht. Tatsächlich wird der Unterschied von R_k zu $f_k(D_k)$ nicht per Hausdorff-Abstand, sondern durch eines der früher diskutierten Verzerrungsmaße bestimmt. Man kann zeigen, daß mit den (partiell definierten) Funktionen f_k die zugehörige mengenwertige Funktion $F(A) = \bigcup_k f_k(A)$ kontrahierend ist und somit die für IFS gemachten Überlegungen auch hier gelten.

Bislang haben wir uns lediglich um Schwarz-Weiß-Bilder in unseren Überlegungen gekümmert. Tatsächlich gilt das Gesagt im wesentlichen auch für Grauwert- bzw. Farbbilder; hier kommt dann noch eine Anpassung der Farbinformation hinzu, so wie die Abbildung f_3 in Bild 23 nicht nur ein Verschieben und Drehen, sondern auch eine Grauwertanpassung beinhaltet. Nach Jacquin wird durch $f_k = m_k \circ g_k$ zerlegt, wobei g_k eine Verschiebung und Skalierung bewirkt und m_k eine Transformation der Masse. Ist –genauer– $T_k = g_k(D_k)$ und bezeichnet t_{ij}, $i, j \in \{0, \ldots, M-1\}$, das (i,j)-te Pixel in T_k, so ist $m_k(t_{ij}) = \alpha_k t_{I_k(i,j)} + \Delta_k)$. $\alpha_k t + \Delta_k$ bezeichnet dabei eine Grauwertanpassung und I_k sei eine Isometrie, die man als Permutation der Pixel innerhalb von T_k auffassen kann; konkret betrachtet man:

1. Drehung um 90 Grad;
2. Drehung um 180 Grad;
3. Drehung um 270 Grad;
4. Spiegelung an vertikaler Spiegelachse;
5. Spiegelung an horizontaler Spiegelachse;
6. Spiegelung an der Diagonalen "links unten bis rechts oben";
7. Spiegelung an der Diagonalen "rechts unten bis links oben";
8. Identität.

Zur Darstellung von R_k benötigt man so g_k, α_k, Δ_k und I_k. In der Praxis schränkt man die Wahl der α_k-Werte noch drastisch ein, und häufig betrachtet man lediglich die Identität für I_k. Dennoch bleibt die Bestimmung der Parameter ein erhebliches Rechenproblem. Interessierte können sich zu diesem Thema recht aktuell unter
`http://www.informatik.uni-leipzig.de/cgip/projects/fic.html`
(der Homepage der "Fraktalgruppe" von Dietmar Saupe) informieren. Dort gibt es u.a. Arbeiten, die zum einen die Schwierigkeit optimaler Blockzuordnung betreffen (Nachweis der NP-Härte), aber auch verschiedendste heuristische Lösungsansätze, in der unterschiedliche auch in dieser Vorlesung besprochene Methoden wie Vektorquantisierung, Fouriertransformation und Adaption in fraktale Komprimierungsschemata eingebaut werden.

9 Transformcodierung und JPEG-Standard

Das allgemeine Konzept der Transformation oder Umformung ist sehr gut in der Mathematik und auch in anderen Gebieten bekannt. Das Schema für die Transformation sieht folgendermaßen aus: Seien $x_1, x_2, \ldots, x_k$ Größen, wie z.B. Zahlen, Vektoren, Funktionen etc. Dann ändern wir diese Objekte mit Hilfe einer Transformation T und bekommen $\theta_1, \theta_2, \ldots, \theta_t$, die möglicherweise ganz anders als p_i aussehen, die man aber einfach weiterverarbeiten kann. Schematisch kann man eine Transformation folgendermaßen darstellen:

$$\begin{tikzcd}
x_1, x_2, \ldots, x_k \arrow[r, "\textit{Transformation}"] & \theta_1, \theta_2, \ldots, \theta_t \arrow[d, "\textit{Bearbeitung}"] \\
\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_k & \hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_t \arrow[l, "\textit{inverse Transformation}"']
\end{tikzcd}$$

Unser Ziel ist es, Transformationen zu finden, sodaß man $\theta_1, \theta_2, \ldots, \theta_t$ effizienter (verlustbehaftet) komprimieren kann als die Folge $x_1, x_2, \ldots, x_k$ direkt. Dann ist $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_t$ eine Rekonstruktion des Komprimierungsverfahrens der Folge $\theta_1, \theta_2, \ldots, \theta_t$, und $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_k$ ist eine Rekonstruktion der Eingabe $x_1, x_2, \ldots, x_k$. Viele der bislang vorgestellten Verfahren lassen sich auch als Transformationscodierungen begreifen.

Dennoch wollen wir das Grundprinzip anhand eines einfachen Beispiels erläutern, nämlich dem schon früher verwendeten, bei dem die Daten aus einer Folge von Paaren (Länge,Gewicht) von Personen bestehen. Ein möglicher Datensatz ist in Tabelle 24 aufgeführt. Betrachtet man Länge und Gewicht als (x, y)-Koordinaten, so sieht man in Bild 24, daß sich die Daten längs einer Geraden der Art $y = 0,5x$ häufen. Drehen wir den Datensatz also durch $\theta = \mathbf{A}\mathbf{x}$, wobei

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

der zweidimensionale Eingabevektor ist und

$$\mathbf{A} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix}$$

die Drehmatrix sowie

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

Größe	Gewicht
163	85
188	94
150	75
175	85
140	65
200	102
170	80
125	55
100	40
125	77
173	74
155	70
190	82
160	60

Tabelle 24: Der Eingangs-Datensatz

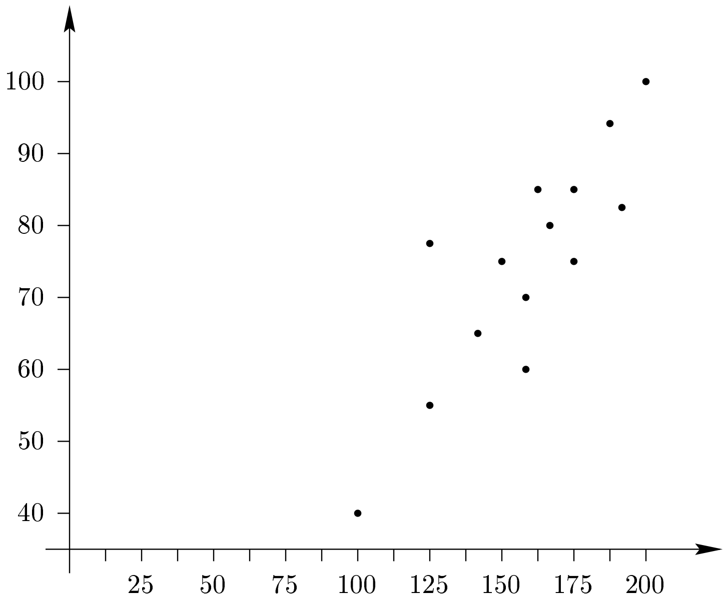


Abbildung 24: Graphische Darstellung der Eingangsdaten

x-Wert	y-Wert
184	3
210	0
168	0
195	-2
154	-4
225	2
188	-4
136	-7
107	-9
146	13
188	-11
170	-7
207	-12
170	-18

Abbildung 25: Die transformierten Werte

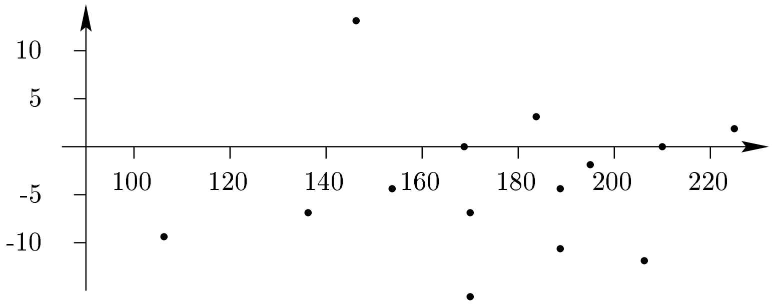


Abbildung 26: Die transformierten Werte

der Vektor der Transformierten; speziell ist hier $\phi = \arctan(0,5) \approx 26,565^o$, d.h.

$$\mathbf{A} = \begin{bmatrix} 0,89442719 & 0,4472136 \\ -0,4472136 & 0,89442719 \end{bmatrix}$$

Auf diese Weise erhält man als Tabelle der transformierten Werte 25, was graphisch der in Bild 26 dargestellten Situation entspricht.

Wir beobachten bei den transformierten Werten, daß sich die "Energie" in der ersten Komponente "ballt", was intuitiv eine (verlustbehaftete) Komprimierung um den Faktor zwei gestattet. Täten wir dies tatsächlich und wendeten auf die "komprimierte Folge" die inverse Transformation, gegeben

Größe	Gewicht
165	82
188	94
150	75
174	87
138	69
201	101
168	84
122	61
96	48
131	65
168	84
152	76
185	93
152	76

Tabelle 25: Die rücktransformierten Werte

durch

$$\mathbf{A}^{-1} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

an, so erhielten wir die Werte aus Tabelle 25.

In diesem Fall gilt:

$$\sum_{i=0}^{N-1} (x_i - \hat{x}_i)^2 = \sum_{i=0}^{N-1} (\theta_i - \hat{\theta}_i)^2$$

wobei

$$\hat{\theta}_i = \begin{cases} \theta_i & i = 0, 2, 4, \ldots \\ 0 & \text{sonst} \end{cases}$$

und $\hat{x}_i$ der x_i entsprechende rekonstruierte Wert ist. Diese Eigenschaft gilt allgemein für Transformationsmatrizen mit der Eigenschaft $A^{-1} = A^T$.

9.1 Die Transformation

Zuerst betrachten wir Transformationen für eindimensionale Folgen, wie zum Beispiel (digitalisierte) Sprache und Audio-Folgen. Es sei $p_0, p_1, p_2, \ldots$ die Eingabefolge. Wir werden folgende lineare Transformationen betrachten: Für jeden Block

$$x_0 = p_\ell,\ x_1 = p_{\ell+1}, \ldots, x_{N-1} = p_{\ell+N-1}$$

der Länge N, mit $\ell = 0, N, 2N, \ldots$, transformieren wir $x_0, x_1, \ldots, x_{N-1}$ folgendermaßen:

$$\theta_n = \sum_{i=0}^{N-1} x_i a_{n,i},$$

wobei die Transformation vollständig durch die Konstanten $a_{n,i}$ definiert wird. Die Länge N hängt von praktischen Anwendungen ab. Es ist klar, daß mit größerem N auch die Transformation komplexer wird. Seien nun x und θ die Vektoren. Dann können wir eine lineare Transformation in Matrixform darstellen: $\theta = Ax$. Die inverse Transformation definiert eine Matrix B, so daß $x = B\theta$ gilt. Die Matrix B ist die Inverse zu A, das heißt $AB = BA = I$. Transformationskodierung ist eine der populärsten Methoden für die Komprimierung von Bildern. Deshalb werden wir bis Ende dieses Kapitels meistens den zweidimensionalen Fall der linearen Transformation betrachten. Wir definieren solche Transformationen für einen gegebenen $N \times N$ Block

$$X = \begin{bmatrix} x_{0,0} & x_{0,1} & \ldots & x_{0,N-1} \\ x_{1,0} & x_{1,1} & \ldots & x_{1,N-1} \\ \vdots & & & \vdots \\ x_{N-1,0} & x_{N-1,1} & \ldots & x_{N-1,N-1} \end{bmatrix}$$

folgendermaßen:

$$\Theta = AXA^T$$

und die inverse Transformation: $X = B\Theta B^T$. Alle Transformationen, die wir hier betrachten werden, sind orthonormal, das heißt, daß die inverse Matrix B (für den reellen Bereich) einfach die transponierte Matrix A^T ist:

$$B = A^{-1} = A^T.$$

Daher ist es einfach, die inverse Transformation zu bekommen:

$$X = A^T \Theta A$$

Die Zeilen der Transformationsmatrix nennen wir **Basisvektoren** und die Elemente nach der Transformation nennen wir oft **Transformationskoeffizienten**.

Man beachte, daß wir auf diese Art und Weise nur sog. separable Transformationen für den zweidimensionalen Fall erhalten; denn die allgemeine Transformationsformel lautet

$$\Theta_{k,\ell} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{i,j} a_{i,j,k,\ell},$$

(dies wäre ein Tensorprodukt) und wir betrachten als Spezialisierung:

$$\Theta_{k,\ell} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{i,j} a_{k,i} a_{\ell,j}.$$

Die Wirksamkeit einer Transformation hängt davon ab,

- wie groß die Reduktion der Korrelation der Eingabepixel (oder Signale) ist und
- wieviel Energiekompaktion die Transformation gibt.

Die Korrelation in einer Eingabefolge ist die Quelle der Redundanz, die in einem effizienten Komprimierungsverfahren entfernt werden soll.

Eine wichtige Eigenschaft der orthonormalen Transformationen ist, daß solche Transformationen die Energie erhalten:

$$\sum x_{i,j}^2 = \sum \theta_{i,j}^2.$$

Die Transformationen 'verschieben nur die Energie in Richtung der DC-Koeffizienten'.

Beispiel Betrachten wir die folgende Transformationsmatrix:

$$A = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

A ist orthonormal, weil

$$\frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^T = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^T \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Die erste Zeile der Matrix: $(1/\sqrt{2}, 1/\sqrt{2})$ enspricht einem 'Tiefpaß'-Signal, daß heißt, es gibt keine Änderung von einer Komponenten zu nächsten und die zweite Zeile: $(1/\sqrt{2}, -1/\sqrt{2})$ enspricht einem 'Hochpaß'-Signal. Betrachten wir eine Folge, in der beide Elemente gleich sind. Nach der Transformation soll dann das zweite Element Null sein. Es sei (α, α) eine solche Eingabefolge. Es gilt:

$$\theta = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{2}\alpha \\ 0 \end{bmatrix}$$

Der erste ('Tiefpaß') Koeffizient heißt Nieder-Frequenz-Koeffizient und der nächste – Hoch-Frequenz-Koeffizient. Für den 2-dimensionalen Fall und für die Eingabematrix $\begin{bmatrix} \alpha & \alpha \\ \alpha & \alpha \end{bmatrix}$ erhalten wir die folgende Transformation:

$$\Theta \;=\; \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} \alpha & \alpha \\ \alpha & \alpha \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^T \;=\; \begin{bmatrix} 2\alpha & 0 \\ 0 & 0 \end{bmatrix}$$

Eine interessante Eigenschaft ist, daß nach der Transformation der oberen Eingaben sich die ganze Energie im Nieder-Frequenz-Koeffizienten konzentriert.

Aus historischen Gründen heißt der $\theta_{0,0}$ Koeffizient **DC-Koeffizient** und die anderen **AC-Koeffizienten**. DC steht für *direct current*, das heißt, daß der Strom unveränderlich in der Zeit ist, wobei AC für *alternating current* steht. Einen Grund für diese Namensgebung sieht man, wenn man eine inverse Transformation für A betrachtet:

$$X \;=\; A\Theta A^T \;=\; \frac{1}{2}\begin{bmatrix} \theta_{0,0}+\theta_{0,1}+\theta_{1,0}+\theta_{1,1} & \theta_{0,0}-\theta_{0,1}+\theta_{1,0}-\theta_{1,1} \\ \theta_{0,0}+\theta_{0,1}-\theta_{1,0}-\theta_{1,1} & \theta_{0,0}-\theta_{0,1}-\theta_{1,0}+\theta_{1,1} \end{bmatrix}$$

Das heißt:

$$X \;=\; \theta_{0,0}\alpha_{0,0}+\theta_{0,1}\alpha_{0,1}+\theta_{1,0}\alpha_{1,0}+\theta_{1,1}\alpha_{1,1},$$

wobei

$$\alpha_{0,0}=\tfrac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \alpha_{0,1}=\tfrac{1}{2}\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

$$\alpha_{1,0}=\tfrac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \quad \alpha_{1,1}=\tfrac{1}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Beobachten Sie, daß alle Elemente der $\alpha_{0,0}$ Matrix gleich sind. Daher kommt die DC Benennung. □

9.2 Spezielle Transformationen für Bildverarbeitung

Wir zeigen die wichtigsten Transformationen, die man für die Bildverarbeitung benutzen kann. Weitere Einzelheiten zu diesem Thema finden Sie z.B. unter [2].

9.2.1 Karhunen-Loéve-Transformation KLT

KLT ist nachweislich optimal hinsichtlich zweiter Kriterien: Es dekorreliert die Daten am besten und packt damit am besten die Information in den niederfrequenten Bereich. Die entsprechenden mathematischen Meßgrößen η_C und η_E werden wir weiter unten einführen.

Die wichtigsten Anfangsschritte der Transformation sind:

1. Berechne Kovarianzmatrix $COV(X)$ der Daten.
2. Bestimme die Eigenvektoren.

Das alles gilt jeweils für jedes Bild oder für jede Bildgruppe, da die Basisvektoren nur für die jeweilige Datenmenge chrakteristisch sind. Zusätzlich müssen noch die Basisvektoren zusammen mit den Kompressionseingaben verschickt werden. Deshalb ist diese Methode in der Praxis nicht nützlich.

Wir geben jetzt die wichtigsten Begriffe für eine Bildtransformation, und dann zeigen wir, wie die Karhunen-Loéve-Transformation aussieht.

Erinnern wir zunächst die Begriffe Mittelwert und Varianz für das ganze Bild ($N \times M$ Pixel):

$$\bar{x} = \frac{1}{N \times M} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{i,j}$$

$$\sigma^2 = \frac{1}{N \times M} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (x_{i,j} - \bar{x})^2.$$

Jetzt betrachten wir –als Einführung für den 2-dim. Fall– die **1-dim. Korrelation**.

Für eine (1-dim.) Folge x ist die Varianz:

$$\sigma^2 = \sigma_{xx}^2 = E[(x - \bar{x})^2].$$

Für zwei Folgen x und y definieren wir als ihre **Kovarianz**

$$\sigma_{xy}^2 = E[(x - \bar{x})(y - \bar{y})].$$

(Hier und im folgenden werden wir als "Erwartungswert" stets den Mittelwert ansetzen, was ja auch ein üblicher Schätzer ist. Ähnlich lässig verfahren wir mit anderen Begriffen aus Wahrscheinlichkeitstheorie und Statistik.) Ist $\sigma_{xy}^2 = 0$, so sind die beiden Folgen **unkorreliert**. Jetzt betrachten wir als Beispiel 3 Folgen: x, y und z. Dann setzen wir:

$$COV(X) = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 \end{bmatrix}$$

Für k Folgen ist $COV(X)$ analog eine $k \times k$-Matrix.

Jetzt definieren wir für die Folge

$$x_1, x_2, \ldots, x_M$$

die Kovarianz

$$\sigma^2_{1,k+1} = \sigma^2_{k+1,1} = \frac{1}{M-k} \sum_{i=1}^{M-k} (x_i - \bar{x})(x_{i+k} - \bar{x}).$$

d.i. die Kovarianz für die Folgen $x_1, x_2, \ldots, x_{M-k}$ und $x_{k+1}, x_{k+2}, \ldots, x_M$, wobei wir als Mittelwert für beide Folgen $\bar{x}$ nehmen. Vereinfachend setzen wir weiter:

$$\sigma^2_{1,k+1} = \sigma^2_{k+1,1} = \frac{1}{M} \sum_{i=1}^{M-k} (x_i - \bar{x})(x_{i+k} - \bar{x}).$$

Für $k \ll M$ ist das immer eine gute Abschätzung. Dann bekommen wir:

$$\sigma^2_{11} = \sigma^2_{22} = \ldots \sigma^2_{KK} = \sigma^2$$

$$\sigma^2_{12} = \sigma^2_{23} = \ldots \sigma^2_{K-1,K} = \sigma^2_1$$

usw., wobei wir $\sigma^2_{\ell,k}$ ähnlich definieren wie $\sigma^2_{1,k+1}$ nur für die Folgen, die mit x_ℓ bzw. $x_{\ell+k}$ starten. Dann ist

$$COV(X) = \begin{bmatrix} \sigma^2 & \sigma^2_1 & \sigma^2_2 & \ldots & \sigma^2_{K-1} \\ \sigma^2_1 & \sigma^2 & \sigma^2_1 & \ldots & \sigma^2_{K-2} \\ \vdots & \vdots & \vdots & & \vdots \\ \sigma^2_{K-1} & \sigma^2_{K-2} & \sigma^2_{K-3} & \ldots & \sigma^2 \end{bmatrix}$$

Wir normalisieren die Matrix:

$$COV(X) = \sigma^2 \begin{bmatrix} 1 & \rho_1 & \rho_2 & \ldots & \rho_{K-1} \\ \rho_1 & 1 & \rho_1 & \ldots & \rho_{K-2} \\ \vdots & \vdots & \vdots & & \vdots \\ \rho_{K-1} & \rho_{K-2} & \rho_{K-3} & \ldots & 1 \end{bmatrix},$$

wobei $\rho_k = \sigma^2_k / \sigma^2$ der **k-te Korrelationskoeffizient** ist. Zur Bildverarbeitung paßt der folgende Fall sehr gut:[18]

$$\rho_k = \rho^k,$$

[18] Genau genommen wird dabei mit der Annahme gearbeitet, die Eingabefolge sei ein stationärer Markov-Prozeß k-ter Ordnung. In der Praxis bewährt sich diese an und für sich falsche Annahme.

mit $\rho = \rho_1$. Wir nennen ρ Zwischen-Element-Korrelation. Dann definieren wir

$$COR(X) = \begin{bmatrix} 1 & \rho & \rho^2 & \dots & \rho^{K-1} \\ \rho & 1 & \rho & \dots & \rho^{K-2} \\ \vdots & \vdots & \vdots & & \vdots \\ \rho^{K-1} & \rho^{K-2} & \rho^{K-3} & \dots & 1 \end{bmatrix}$$

Man vergleiche die gemachten Ausführungen auch mit denen in Unterabschnitt 8.2 zum Thema Autokorrelation.

Für den 2-dim. Fall verallgemeinern wir diese Begriffe zu vertikalen und horizontalen Folgen. Die Verallgemeinerung ist nicht trivial. Daher betrachten wir jetzt nur (noch) den eindimensionalen Fall.

Es seien $COV(X) = (X_{i,j})$ und $COV(Y) = (Y_{i,j})$ die Kovarianz-Matrizen für eine eindimensionale Folge und für ihre Transformation. Dann setzen wir:

$$\Sigma X = \sum_{\substack{1 \le i,j \le N \\ i \ne j}} |X_{i,j}|$$

und

$$\Sigma Y = \sum_{\substack{1 \le i,j \le N \\ i \ne j}} |Y_{i,j}|$$

und definieren damit die **Dekorrelationswirkung** η_C folgendermaßen:

$$\eta_C = 1 - \frac{\Sigma Y}{\Sigma X}.$$

Eine andere Größe, die die Qualität einer Transformation beschreibt, ist der **Energie-Packung**s-Koeffizient in den ersten M von N diagonalen Komponenten:

$$\eta_E = \frac{\sum_{j=1,k=j}^{M} Y_{j,k}}{\sum_{j=1,k=j}^{N} Y_{j,k}}.$$

Die Zeilen der Karhunen-Loéve-Transformation sind die Eigenvektoren der Autokorrelationsmatrix. Für jedes $i = 0, 1, \dots, N-1$ lösen wir folgende Gleichung

$$\tan N\omega_i = \frac{-(1-\rho^2)\sin\omega_i}{(1+\rho^2)\cos\omega_i - 2\rho},$$

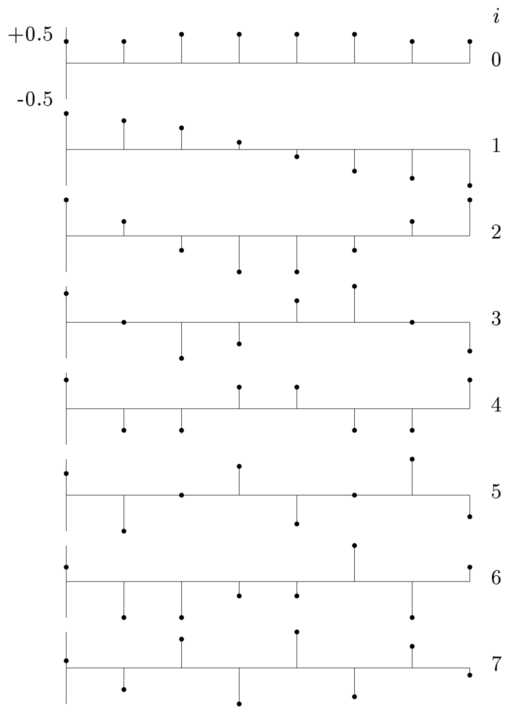


Abbildung 27: Die acht Basisvektoren der KLT für $\rho = 0,91$

wobei ρ die Zwischen-Element-Korrelation ist, und dann benutzen wir die Lösungen, um die Eigenvektoren zu berechnen:

$$\lambda_i = \frac{1-\rho^2}{1+\rho^2-2\rho\cos\omega_i}.$$

Basisvektoren für $0 \leq i, j \leq N-1$ sind dann:

$$a_{i,j} = \left(\frac{2}{N+\lambda_i}\right)^{1/2} \sin\left[\omega_i\left(j-\frac{N-1}{2}\right)+\frac{(i+1)\pi}{2}\right]$$

In Bild 27 sehen Sie die Werte der Basisvektoren für $N = 8$ und $\rho = 0,91$.

9.2.2 Diskrete Fouriertransformation

Diese ist die einzige komplexzahlige Transformation. Wir hatten sie schon kurz im vorigen Kapitel betrachtet.

$$a_{\ell,k} = \sqrt{1/N}\left(\cos\frac{2\pi\ell k}{N} - i\sin\frac{2\pi\ell k}{N}\right).$$

Die Basisvektoren für $N = 8$ sehen folgendermaßen aus: Die reelle Basis:

$$\begin{bmatrix} 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 \\ 0.35 & 0.25 & 0.00 & -0.25 & -0.35 & -0.25 & 0.00 & 0.25 \\ 0.35 & 0.00 & -0.35 & 0.00 & 0.35 & 0.00 & -0.35 & 0.00 \\ 0.35 & -0.25 & 0.00 & 0.25 & -0.35 & 0.25 & 0.00 & -0.25 \\ 0.35 & -0.35 & 0.35 & -0.35 & 0.35 & -0.35 & 0.35 & -0.35 \\ 0.35 & -0.25 & 0.00 & 0.25 & -0.35 & 0.25 & 0.00 & -0.25 \\ 0.35 & 0.00 & -0.35 & 0.00 & 0.35 & 0.00 & -0.35 & 0.00 \\ 0.35 & 0.25 & 0.00 & -0.25 & -0.35 & -0.25 & 0.00 & 0.25 \end{bmatrix}$$

und die imaginäre Basis

$$\begin{bmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.25 & 0.35 & 0.25 & 0.00 & -0.25 & -0.35 & -0.25 \\ 0.00 & 0.35 & 0.00 & -0.35 & 0.00 & 0.35 & 0.00 & -0.35 \\ 0.00 & 0.25 & -0.35 & 0.25 & 0.00 & -0.25 & 0.35 & -0.25 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & -0.25 & 0.35 & -0.25 & 0.00 & 0.25 & -0.35 & 0.25 \\ 0.00 & -0.35 & 0.00 & 0.35 & 0.00 & -0.35 & 0.00 & 0.35 \\ 0.00 & -0.25 & -0.35 & -0.25 & 0.00 & 0.25 & 0.35 & 0.25 \end{bmatrix}$$

Es scheint so zu sein, daß man den imaginären Anteil nicht einfach vernachlässigen kann, so wie dies in der Praxis wohl manchmal passiert. Das Problem der "Datenaufblähung" durch Einführung eines Imaginärteils umgeht die diskrete Cosinustransformation:

9.2.3 DCT – Diskrete Cosinus-Transformation

$$a_{i,j} = \begin{cases} \sqrt{1/N} & i = 0,\ j = 0, 1, \ldots, N-1 \\ \sqrt{2/N} \cos \frac{(2j+1)i\pi}{2N} & i = 1, \ldots, N-1,\ j = 0, 1, \ldots, N-1. \end{cases}$$

Für den 8×8-Fall ergeben sich als Basisvektoren:

$$A = \begin{bmatrix} 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 \\ 0.49 & 0.42 & 0.28 & 0.10 & -0.10 & -0.28 & -0.42 & -0.49 \\ 0.46 & 0.19 & -0.19 & -0.46 & -0.46 & -0.19 & 0.19 & 0.46 \\ 0.42 & -0.10 & -0.49 & -0.28 & 0.28 & 0.49 & 0.10 & -0.42 \\ 0.35 & -0.35 & -0.35 & 0.35 & 0.35 & -0.35 & -0.35 & 0.35 \\ 0.28 & -0.49 & 0.10 & 0.42 & -0.42 & -0.10 & 0.49 & -0.28 \\ 0.19 & -0.46 & 0.46 & -0.19 & -0.19 & 0.46 & -0.46 & 0.19 \\ 0.10 & -0.28 & 0.42 & -0.49 & 0.49 & -0.42 & 0.28 & -0.10 \end{bmatrix}$$

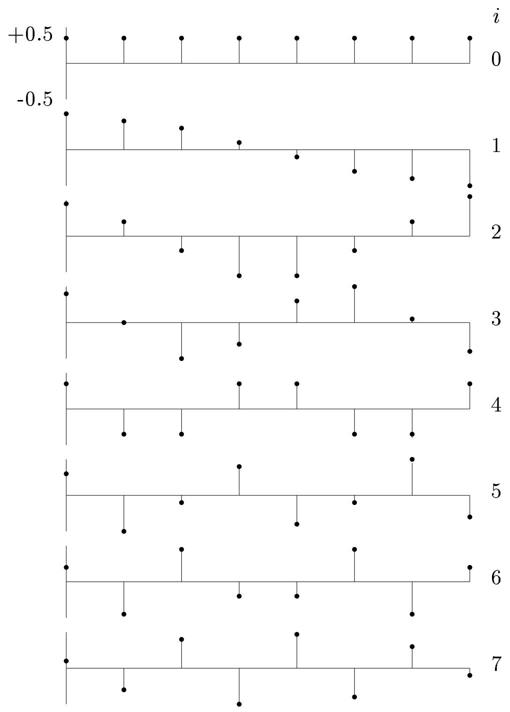


Abbildung 28: Die acht Basisvektoren der DCT

Ihre graphische Darstellung entnehmen Sie bitte Bild 28.

Vergleichen Sie die Werte der Basisvektoren der diskreten Cosinus-Transformation mit den Werten für KLT in Bild 27.

Wir betrachten jetzt drei Beispiele:

1. Zwei aneinanderstoßende Flächen

$$X = \begin{bmatrix} 10 & 10 & 10 & 10 & 128 & 128 & 128 & 128 \\ 10 & 10 & 10 & 10 & 128 & 128 & 128 & 128 \\ 10 & 10 & 10 & 10 & 128 & 128 & 128 & 128 \\ 10 & 10 & 10 & 10 & 128 & 128 & 128 & 128 \\ 10 & 10 & 10 & 10 & 128 & 128 & 128 & 128 \\ 10 & 10 & 10 & 10 & 128 & 128 & 128 & 128 \\ 10 & 10 & 10 & 10 & 128 & 128 & 128 & 128 \\ 10 & 10 & 10 & 10 & 128 & 128 & 128 & 128 \end{bmatrix}$$

besitzen folgende Cosinus-Transformierte:

$$\Theta = \begin{bmatrix} 552.0 & -427.7 & 0.0 & 150.2 & 0.0 & -100.4 & 0.0 & 85.1 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

2. Ein Block des Bildes 'Brücke'

$$X = \begin{bmatrix} 94 & 96 & 113 & 135 & 196 & 116 & 110 & 106 \\ 91 & 125 & 120 & 153 & 192 & 135 & 108 & 124 \\ 87 & 119 & 144 & 122 & 190 & 131 & 115 & 132 \\ 88 & 93 & 141 & 100 & 168 & 149 & 128 & 122 \\ 95 & 93 & 139 & 169 & 172 & 147 & 154 & 135 \\ 87 & 100 & 124 & 173 & 180 & 132 & 173 & 178 \\ 104 & 85 & 112 & 120 & 167 & 158 & 132 & 166 \\ 112 & 83 & 124 & 123 & 154 & 152 & 148 & 165 \end{bmatrix}$$

wird transformiert in:

$$\Theta = \begin{bmatrix} 1049.9 & -125.7 & -121.2 & 4.7 & 50.1 & -40.0 & 2.4 & 55.2 \\ -30.4 & 59.0 & -48.7 & 6.0 & -0.2 & -35.8 & -22.7 & 13.3 \\ -17.2 & 5.7 & 12.8 & 16.4 & 24.6 & -11.0 & 0.6 & -5.4 \\ 15.7 & -14.8 & -17.5 & -9.4 & 25.7 & 26.1 & -30.5 & -10.9 \\ -19.9 & 5.6 & -0.6 & 14.8 & -13.1 & 22.2 & 14.5 & 12.3 \\ -30.6 & -3.6 & 13.5 & 17.2 & -8.3 & -18.8 & 20.3 & 19.7 \\ 11.1 & -6.8 & 2.1 & -19.6 & 4.3 & 5.6 & -16.0 & 13.4 \\ 0.5 & 9.5 & -7.4 & 2.8 & 4.0 & 1.0 & 5.5 & -0.2 \end{bmatrix}$$

8	7	5	3	1	1	0	0
7	5	3	2	1	0	0	0
4	3	2	1	1	0	0	0
3	3	2	1	1	0	0	0
2	1	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabelle 26: Typische Bitverteilung für eine 8×8-Transformation

3. Ein einzelner Punkt

$$X = \begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 128 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{bmatrix}$$

ergibt:

$$\Theta = \begin{bmatrix} 94.8 & 4.1 & -19.3 & -11.6 & 14.8 & 17.3 & -8.0 & -20.5 \\ 4.1 & 1.1 & -5.3 & -3.2 & 4.1 & 4.8 & -2.2 & -5.6 \\ -19.3 & -5.3 & 25.2 & 15.1 & -19.3 & -22.7 & 10.4 & 26.7 \\ -11.6 & -3.2 & 15.1 & 9.1 & -11.6 & -13.6 & 6.3 & 16.1 \\ 14.8 & 4.1 & -19.3 & -11.6 & 14.8 & 17.3 & -8.0 & -20.5 \\ 17.3 & 4.8 & -22.7 & -13.6 & 17.3 & 20.4 & -9.4 & -24.1 \\ -8.0 & -2.2 & 10.4 & 6.3 & -8.0 & -9.4 & 4.3 & 11.1 \\ -20.5 & -5.6 & 26.7 & 16.1 & -20.5 & -24.1 & 11.1 & 28.4 \end{bmatrix}$$

Diskrete Cosinus-Transformation und diskrete Fourier-Transformation hängen eng zusammen: Spiegelt man die N-Punkt-Folge einer DFT am rechten Rand, so erhält man die "zugehörige" $2N$-Punkt-Folge der DCT.

9.3 Bit-Verteilung

Wie aus den Beispielen ersichtlich, ist der Informationsgehalt der Einträge der transformierten Matrix unterschiedlich; ähnlich wie bei der zuvor betrachteten Teilbandkompression massiert sich die Energie im linken oberen Eck, und das ist ja auch eines der Kriterien für eine gute Transformation. Das bedeutet wiederum, daß es sinnvoll ist, bei der quantisierten

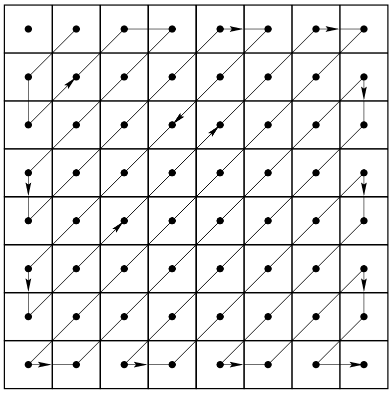


Abbildung 29: Zickzack-Abtastung eines 8×8-Musters

Übertragung dieser transformierten Koeffizienten unterschiedlich viele Bits zu verwenden. Diese Beobachtung führt zur Anwendung des Bitverteilungsalgorithmus aus Unterabschnitt 8.5.3 und wird in diesem Kontext "zonal sampling" genannt. Tabelle 26 zeigt solch eine typische Bitverteilung.

Chen und Pratt haben ein anderes Bitverteilungsverfahren vorgeschlagen, bei dem die zu übertragenden Werte des transformierten Bildes zunächst quantisiert werden (mit einem Quantisierer, der u.a. Null als Repräsentanten eines um den Nullpunkt symmetrischen Intervalls hat) und dann in der in Bild 29 angegebenen Weise zickzackartig abgetastet werden. Man kann annehmen, daß die sich so ergebene abgetastete Zahlenfolge ab einem gewissen Index konstant Null ist (dieser Annahme liegt ja offenkundig die in Tab. 26 angegebene fixe Bitverteilung zugrunde); daher erscheint es sinnvoll, eine solche konstante Nullfolge durch ein Sonderzeichen, hier EOB (end of block) genannt, dem Empfänger anzuzeigen. Der Vorteil dieses auch in JPEG verwendeten Schemas liegt darin, daß evtl. noch vorhandene "hochfrequente" Informationen nicht automatisch vernachlässigt werden.[19]

9.4 JPEG

Wir haben jetzt das nötige Rüstzeug zusammen, um den von der JPEG vorgeschlagenen Standard verstehen zu können.

Die wichtigsten Schritte der JPEG (schematisch dargestellt in Bild 30) sind folgende:

- Diskrete Cosinus-Transformation.

[19] Einer ähnlichen Idee waren wir ja schon in Shapiros EZW begegnet.

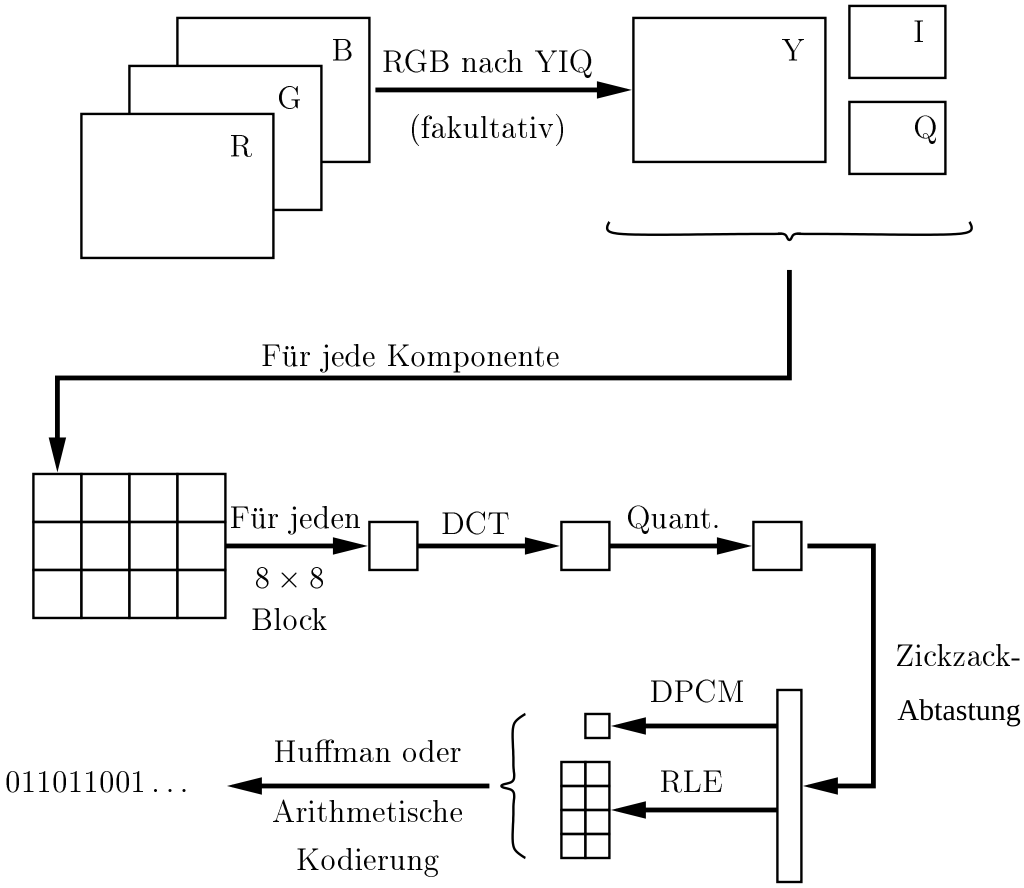


Abbildung 30: Das Schema von JPEG

124	125	122	120	122	119	117	118
121	121	120	119	119	120	120	118
126	124	123	122	121	121	120	120
124	124	125	125	126	125	124	124
127	127	128	129	130	128	127	125
143	142	143	142	140	139	139	139
150	148	152	152	152	152	150	151
156	159	158	155	158	158	157	156

Tabelle 27: Ein 8×8-Block aus dem Sena-Bild...

- Quantisierung.
- Differenz-Codierung auf DC-Komponenten.
- Zickzack-Abtastung der AC-Komponenten.
- Lauflängen-Codierung auf AC-Komponenten.
- Huffman-Codierung oder arithmetische Codierung.

Schauen wir uns die einzelnen Schritte des Schemas 30 genauer an:

1. Der erste Schritt ist fakultativ: Eine Umwandlung von RGB nach YIQ.

   RGB und YIQ sind 3-D Vektorräume, die die Farben und die Helligkeit beschreiben. RGB hat ein passendes Format für die Hardware, wobei YIQ eher der menschlichen Wahrnehmung entspricht. 'I' stellt eine Balance zwischen orange und cyan dar. 'Q'-Komponente beschreibt die Balance zwischen grün und magenta. 'I' und 'Q' zusammen zeigen die Farbe jedes Pixels. 'Y'-Komponenete stellt die Helligkeit des Pixels dar. Um eine solche Umwandlungen zu bekommen, muß man entscheiden, wieviele Bits den einzelnen Komponenten zukommt. Im allgemeinen Fall geben wir viermal mehr Bits für die 'Y'- als für die 'I'- und 'Q'-Komponente; denn die Augen sind empfindlicher für Änderungen der Helligkeit als der Farbe. Weitere Informationen zu Farbformaten finden Sie in Abschnitt 10.1.2

2. Teile das Bild in 8×8 Pixelblöcke.

3. Führe DCT für jeden Block aus. (Dabei wird zuvörderst jeder Pixelwert von 0 bis $2^P - 1$ durch Subtraktion von 2^{P-1} in einen um den Nullpunkt symmetrischen Bereich abgebildet.) Man beobachte das Beispiel eines Blocks aus dem Sena-Bild in den Tabellen 27 und 28.

   Warum DCT und nicht DFT? Die wichtigste Unterschiede zwischen DCT und DFT sind folgende:

39.88	6.56	-2.24	1.22	-0.37	-1.08	0.79	1.13
-102.43	4.56	2.26	1.12	0.35	-0.63	-1.05	-0.48
37.77	1.31	1.77	0.25	-1.50	-2.21	-0.10	0.23
-5.67	2.24	-1.32	-0.81	1.41	0.22	-0.13	0.17
-3.37	-0.74	-1.75	0.77	-0.62	-2.65	-1.30	0.76
5.98	-0.31	-0.45	-0.77	1.99	-0.26	1.46	0.00
3.97	5.52	2.39	-0.55	-0.051	-0.84	-0.52	-0.13
-3.43	0.51	-1.07	0.87	0.96	0.09	0.33	0.01

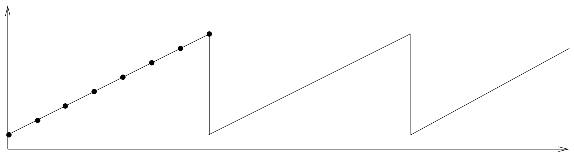
Tabelle 28: liefert diese DCT-Koeffizienten nach JPEG.

(a) DFT generiert komplexe Zahlen, während DCT nur reelle Zahlen ausgibt.

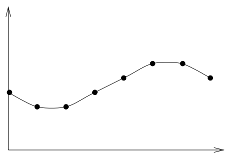
(b) DFT nimmt an, daß die Funktion, die aus $x_0, x_1, \ldots, x_{N-1}$ rekonstruiert wird, periodisch ist (mit Periode N). So betrachtet DFT die Folge

$$8, 16, 24, 32, 40, 48, 56, 64$$

als die Werte der folgenden periodischen Funktion:



Die inverse Transformation für die 'bearbeiteten Werte' des Vektores Ax (z.B. nach Quantisierung) gibt daher die Werte der folgenden Funktion:



DCT hat solche Eigenschaft nicht. Wir betrachten DCT als die ersten N reellen Werte der DFT für die Folge:

$$x_0, x_1, \ldots, x_{N-1}, x_{N-1}, \ldots, x_1, x_0$$

und deshalb entfernen wir diese Nichtstetigkeiten.

(c) Die Komplexität der DCT ist höher als DFT.

Natürlich ist Punkt (a) kein Problem und die entscheidenen Unterschiede sind (b) und (c). JPEG-Kommittee hat DCT gewählt wegen (b) und trotz (c). Man kann empirisch zeigen, daß die Pixels eines Bildes nicht einer periodischen Funktion entsprechen.

In der Praxis benutzt die Software Festpunktarithmetik. Der Weltrekord für die schnellste Ausführung der DCT lautet: 11 Multiplikationen und 29

2	1	0	0	0	0	0	0
-9	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabelle 29: Die Matrix $\ell_{i,j}$ der Quantisierlabels des Sena-Blocks...

Additionen [C. Loeffler, A. Ligtenberg and G. Moschytz, *Practical Fast 1-D DCT Algorithms with 11 Multiplications*, Proc. Int'l. Conf. on Acoustics, Speech, and Signal Processing 1989 (ICASSP '89), pp. 988-991].

4. Quantisierung

   Der Quantisierungsfehler ist die Hauptquelle der Verzerrung in JPEG. Das Verfahren benutzt eine 8×8 Quantisiermatrix Q und das heißt, daß für jeden 8×8-Block X JPEG die Werte

   $$\ell_{i,j} = \lfloor \Theta_{i,j}/Q_{i,j} + 0,5 \rfloor$$

   berechnet, wobei Θ die Matrix X nach der diskreten Cosinus-Transformation ist. Die übliche JPEG-Quantisiermatrix Q sieht folgendermaßen aus:

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

   Im Beispiel in Tabelle 28 ist $\Theta_{0,0} = 39,88$, d.h.

   $$\ell_{0,0} = \lfloor \frac{39,88}{16} + 0,5 \rfloor = \lfloor 2,9925 \rfloor = 2.$$

   Der zugehörige Repräsentant ist daher $32 = 16 * 2$, und der Quantisierfehler wäre hier $7,88$. Die weiteren Daten sind in Tabelle 29 abgelegt; die Matrix der zugehörigen Repräsentanten ist in Tabelle 30 aufgeführt.

32	11	0	0	0	0	0	0
-108	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabelle 30: ...und die zugehörigen Repräsentanten

JPEG definiert 2 Default-Quantisiermatrizen: eine für die Helligkeit (diese sehen Sie oben), daß heißt die Y-Komponente, und eine für die Farben, das heißt für die I- und Q-Komponenete.

Es ist möglich, in JPEG eine eigene Quantisiermatrix definieren. Dann fügt man diese Matrix in den Kopf des kodierten Bildes.

5. DPCM auf DC-Komponenten

   DC-Komponenten sind groß und haben unterschiedliche Werte. Man kann aber sehen, daß viele DC-Komponenten ähnliche Werte wie seine Vorgänger haben. Deshalb wählt man anstelle der direkten Codierung der DC-Komponenten ein DPCM-Verfahren: JPEG codiert die Differenzen zwischen Nachbarkomponenten.

   Die Differenzen komprimiert man mit Hilfe des Huffman-Verfahrens oder der arithmetischen Codierung.

6. Zickzack-Abtastung der AC-Komponenten

   Hierdurch wird der 2-D Block in einen 1-D Vektor umgewandelt, so daß diese Konversion die Energie des Blockes im Anfang des Vektores bündelt.

7. *Lauflängencodierung* auf AC-Komponenten

   Man kann sehen, daß viele AC-Komponenten Null sind. JPEG codiert diese Werte als Paare

   $$(skip, value),$$

   wobei *skip* die Anzahl der Nullen und *value* die nächste Nicht-Null-Komponente ist. $(0, 0)$ steht für das Ende. Dann komprimiert JPEG diese Folgen mit Hilfe des Huffman-Verfahrens oder der arithmetischen Codierung.

Weitere Einzelheiten des JPEG-Verfahrens finden Sie z.B. in [9, 10].

123	122	122	121	120	120	119	119
121	121	121	120	119	118	118	118
121	121	120	119	119	118	117	117
124	124	123	122	122	121	120	120
130	130	129	129	128	128	128	127
141	141	140	140	139	138	138	137
152	152	151	151	150	149	149	148
159	159	158	157	157	156	155	155

Tabelle 31: Rekonstruktion des Sena-Blocks

Die rekonstruierten Werte für den Sena-Block finden Sie in Tabelle 31.

Wer Lust bekommen hat, etwas mit JPEG herumzuexperimentieren, der kann einmal die Applets unter
`http://www.cs.sfu.ca/undergrad/CourseMaterials/CMPT479/material/misc/interactive_jpeg/Ijpeg.html`
anschauen.

Es sei abschließend erwähnt, daß es auch eine JPEG-Variante gibt, die die fortschreitende Bildübertragung unterstützt, s. [12].

9.5 Ein Vergleich von Verfahren zur Bildkompression

"Ein Bild sagt mehr als tausend Worte" beschreibt treffend den Wert visueller Information für den Menschen. Leider umfaßt ein hochaufgelöstes Digitalbild auch weit mehr Datenbits als nämliche tausend Worte, was die Notwendigkeit des Einsatzes von Kompressionsverfahren unterstreicht. Manche Anwendungen, z.B. im medizinischen oder militärischen Bereich erfordern eine getreue Wiedergabe des (digitalisierten) Originalbildes, da verlustbehaftete Kompressionsverfahren dort nicht tolerierbare Störungen mit sich bringen ("Ist jener Fleck dort ein kleines Geschwür oder ein Artefakt des Kompressionsverfahrens?") In WWW-Anwendungen sind typischerweise gewisse Verzerrungen tolerierbar (z.B. in elektronischen Zeitungen); als Kompromiß bietet sich die fortschreitende Bildübertragung an.

Prädikative Methoden (wie Differentialcodierung oder verlustfreier JPEG-Standard), bei denen im wesentlichen codierte Differenzen des aktuellen Pixels vom "erwarteten Pixel" (das aus der Kenntnis benachbarter Pixel errechnet wird) übertragen werden, sind im wesentlichen in Anwendungen im Einsatz, wo nur geringe Störungen toleriert werden können und dafür geringe Kompressionsfaktoren annehmbar sind.

Für mittlere Kompressionsraten sind Umformungs- und Teilbandcodierungen sehr geeignet. Umformungsbasierte Kompressionsverfahren wie JPEG arbeiten auf Datenblöcken und zeigen daher bei hohen Kompressionsrändern Artefakte an den Blockrändern (blocking effect), welche Multiresolutionsansätze nicht aufweisen. Besonders deutlich erscheint der Blockeffekt in "ruhigen" Bereichen. Andererseits neigen Teilbandcodierungen dazu, aufgrund der nicht "perfekten" Filter Welleneffekte (ringing effect) an Rändern aufzuweisen. Durch geeignete Abstimmung der Filter aufeinander läßt sich dieser Effekt jedoch minimieren. Durch Techniken wie bei EZW läßt sich eine fortschreitende Bildübertragung erreichen.

Fraktale Kompressionsverfahren zeigen gute Leistungen bei relativ hohen Kompressionsraten (70 bis 80), die Bilder erscheinen dann aber oft "verschmiert" im Vergleich zum Original. Des weiteren erfordern sie einen hohen Rechenaufwand auf Seiten des Codierers und sind daher eher für "asymmetrische Anwendungen" wie im Multimediabereich geeignet. Ähnlich einzuordnen sind dieser Tage (in dieser Vorlesung gar nicht besprochene) Kompressionsverfahren "der zweiten Generation", bei denen der Codierer zunächst versucht, gewisse Strukturen innerhalb des Bildes zu finden und dann diese Information zu übertragen. Auch diese Verfahren zeigen bei hohen Kompressionsraten sehr gute Bildqualitäten.

Wer hierzu vergleichende Bilder studieren will, sei auf [3, 12] verwiesen.

10 Zwei weitere Anwendungen

In diesem Abschnitt wollen wir zwei Anwendungsgebiete der Datenkompression behandeln, die bislang vernachlässigt wurden: die Kompression von Bewegtbildern am Beispiel des MPEG-Standards und —sehr knapp— die Kompression von Audio-Daten.

10.1 Allgemeines zu Bewegtbildern und deren Kompression

Bewegtbilder kann man im wesentlichen wie Standbilder komprimieren, nur daß eben noch die Zeit als weitere Dimension erscheint. Durch prädikative Ansätze wie bei der Differentialcodierung kann man ausnutzen, daß aufeinanderfolgende Bilder i.d.R. einander “ähnlich” sind. Natürlich wird man auch weiterhin die Ähnlichkeiten innerhalb einzelner Bilder ausnutzen wollen; dabei ist zu beachten, daß dem menschlichen Auge Störungen in “ruhigen Zonen” bei Bewegtbildern schlimmer als bei Standbildern erscheinen, während umgekehrt Kantenstörungen bei Standbildern stärker als bei Bewegtbildern wahrgenommen werden.

Ferner hat man bei Bewegtbildern häufig sehr unterschiedliche Anforderungen seitens der Anwendungen vorzuliegen: Bei Ein-Weg-Kommunikation –der heute wohl immer noch verbreitetsten Form der Bewegtbildnutzung– sind asymmetrische Verfahren denkbar, während bei der Zwei-Weg-Kommunikation (Bildtelephon, Videokonferenz,...) schnelle symmetrische Algorithmen gefordert sind.

10.1.1 Bewegungskompensation

Wesentlich bei allen Kompressionsverfahren für Bewegtbilder ist die “Bewegungskompensation”, um die zeitliche Abhängigkeit aufeinanderfolgender Bilder auszunutzen: Wie man in Abb. 31, 32 sieht, erscheint eine pixelweise Differenz von Bildern keine gute Dekorrelation zu liefern, denn Objekte (wie der Kopf und das Dreieck im Beispiel) bewegen sich unterschiedlich durch Szenen, und diese Bewegung muß erkannt und modelliert werden.

Algorithmisch wird dies Problem durch die sog. blockbasierte Bewegungskompensation, s. Abb. 33 angegangen. Das zu codierende Bild wird in $M \times M$ Pixelblöcke unterteilt. Im “früheren” Bild wird ein “möglichst ähnlicher Block” —Differenzen zwischen Blöcken werden z.B. durch MSE gemessen— gesucht und, falls der Fehler einen vorgegebenen Schwellwert unterschreitet, dann wird eine “Fehlerkorrektur” nebst dem “Bewegungsvektor” übertragen; bei Überschreiten des Schwellwertes wird der Block direkt

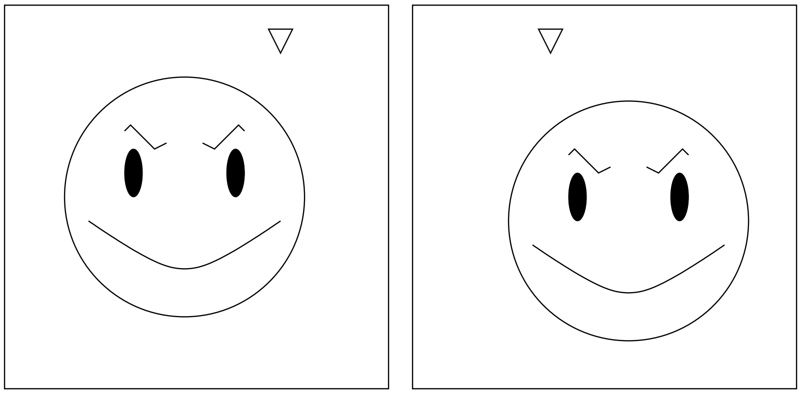


Abbildung 31: Zwei aufeinanderfolgende Bilder

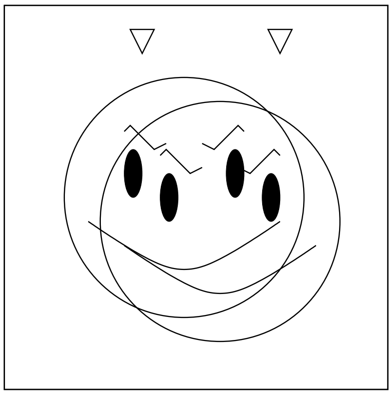


Abbildung 32: überlagert

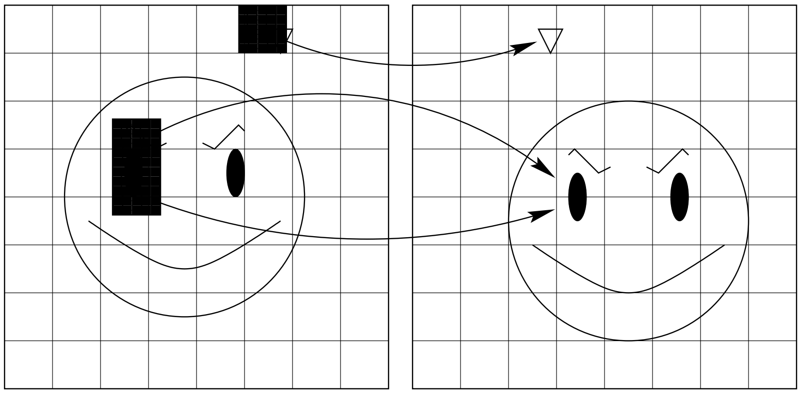


Abbildung 33: Blockbasierte Bewegungskompensation

codiert. Der Suchaufwand zum Auffinden “ähnlicher Blöcke” ist erheblich, wenn auch kleiner als bei fraktaler Kompression (vielleicht vergleichbar mit Vektorquantisierung). Man beachte, daß nicht nur die Blöcke im “früheren Bild” untersucht und verglichen werden, sondern auch feinere Verschiebungen, wie in Abb. 33 ersichtlich.

10.1.2 Farbbildformate

Bekanntermaßen läßt sich ein Farbbild aus den drei Grundfarben Rot, Grün und Blau zusammensetzen (RGB-Format). Dieses Format ist von Farbfernsehern bzw. Computermonitoren sowie Scannern her bekannt. Ebenso benutzt das menschliche Auge diese drei Farbkomponenten zur Farbanalyse. Farbdrucker arbeiten hingegen mit den Komplementärfarben Cyan (ein helles Blau, komplementär zu Rot), Magenta (ein Purpurton, komplementär zu Grün) und Gelb (komplementär zu Blau) (CMY-Format), wobei häufig noch Schwarz hinzugenommen wird, das sich nur theoretisch aus den CMY-Komponenten ergibt (CMYK-Format). Während im Computermonitor ein Farbton additiv aus den RGB-Grundfarben gemischt wird –so ergibt Rot plus Grün ein Gelb–, entsteht der Cyan-Eindruck beim Drucker durch Herausfiltern des Rot-Anteils aus dem Farblichtspektrum, so daß beispielsweise ein Überlagern von Rot und Grün eine Art Dunkelbraun liefert.[20]

Historisch gab es bei der Einführung des Farbfernsehens ein weiteres Problem: Viele Leute besaßen bereits Schwarz-Weiß-Fernseher, und um diese weiterverwenden zu können, wurde das RGB-Signal in Form des YUV-Signals übertragen, und zwar (bei NTCS, ähnlich bei SECAM und PAL) gemäß der Formeln:

$$\begin{aligned} Y &= 0,229R + 0,587G + 0,114B \\ U &= B - Y \\ V &= R - Y \end{aligned}$$

Y beschreibt dabei die Luminanz (Helligkeit) des Bildes, und nur das Luminanzsignal wird von einem Schwarz-Weiß-Gerät umgesetzt. Für die menschliche Wahrnehmung ist das Y-Signal bedeutender als die U- und V-Signale (Chrominanzsignale).

Obschon es in manchen Details unterschiedliche Farbfernsehformate gibt, hat man sich auf ein einheitliches Format für Digitalfernsehen geeinigt (CCIR 601); eigentlich ist es eine ganze Schar von Formaten. Grundfrequenz ist

[20] Abgesehen davon ist es nicht klar, was “genau” Rot z.B. ist, weshalb letztlich der Ausdruck eines eingescannten Farbbildes durchaus anders als das Original erscheinen kann, s. [5].

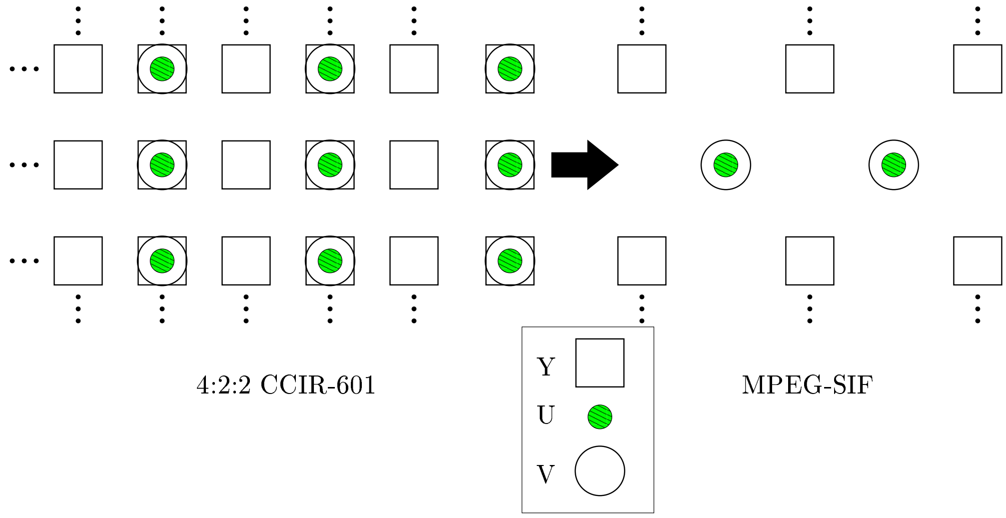


Abbildung 34: Zwei Digitale Bildformate

stets $3,375$ MHz. Beliebtestes Format ist 4:2:2, d.h., das Luminanzsignal wird mit $4*3,375 = 13,5$ MHz und die Chrominanzsignale mit je $2*3,375 = 6,75$ Mhz übertragen. Anvisiert wird eine Übertragung von 30 Bildern je Sekunde zu 720×480 Bildpunkten. Wie in Abb. 34 erkennbar, wird für jeden Bildpunkt ein Luminanzsignal und für jeden zweiten noch zwei Chrominanzsignale erzeugt. Diesen werden (normalisiert und quantisiert) mit je einem Byte übertragen, so daß $60 \times 720 \times 480$ Bytes, also ca. 2 MB, je Sekunde zu übertragen sind. Um weichere Bildübergänge zu erzielen, wird jedes Bild in zwei Durchgängen übertragen: zuerst die geradzahlig und dann die ungeradzahlig numerierten Zeilen.

Das in Abb. 34 dargestellte MPEG-SIF-Format (für MPEG-1 Standard) viertelt die zu übertragende Datenmenge in der angedeuteten Weise. Für Videokonferenzen haben sich noch gröbere Auflösungen (ähnlich aufgebaut wie MPEG-SIF) als Standards durchgesetzt (wieder 30 Bilder je Sekunde):

CIF	Luminanz 288×352	Chrominanz 144×176
QCIF	Luminanz 144×176	Chrominanz 72×88

10.1.3 Videokonferenz

Wir erklären im folgenden den auf dem [Q]CIF-Bildformat beruhenden ITU-T H.261 Standard für Videokonferenzen; im übrigen ist der MPEG-1 Standard sehr ähnlich (s.u.). Das Grundschema der Komprimierung ist in Abb. 35

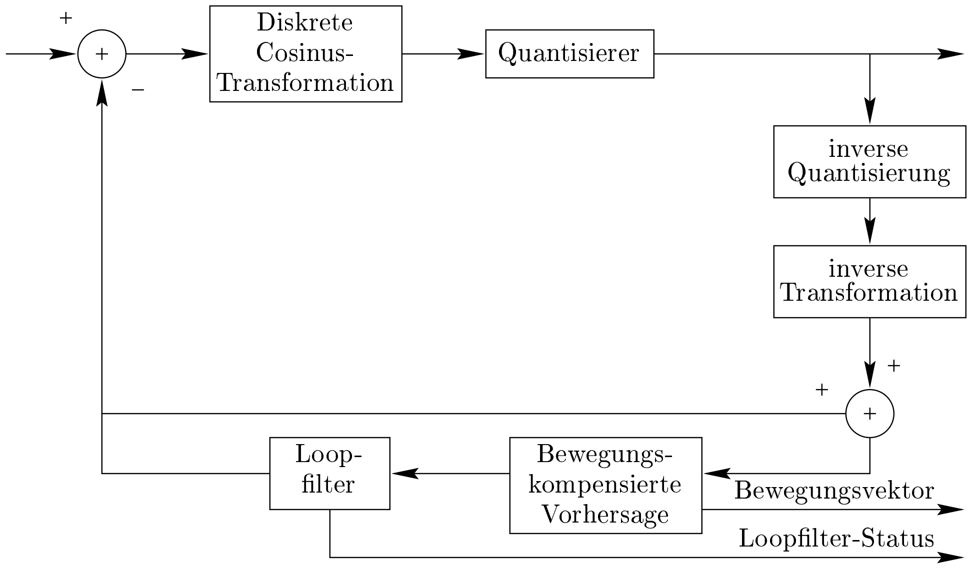


Abbildung 35: Der ITU-T H.261 Codierer

erläutert. Die Dekomprimierung ist als Sub-Algorithmus im Komprimierer enthalten, da dieser prädikativ arbeitet.

Angesichts der Zielanwendung sollten sowohl Codierung als auch Decodierung sehr schnell gehen (Echtzeit); das gewählte Bildformat gewährleistet an und für sich schon eine relativ geringe Bitrate.

Am rechenintensivsten erscheint prinzipiell die schon oben diskutierte Bewegungskompensation. Wie begegnet der ITU-T H.261 Standard diesem Problem?

Der Luminanz- bzw. die Chrominanzanteile eines Bildes werden jeweils in 8×8 große Blöcke unterteilt. Ein Macroblock besteht aus 4 Luminanzblöcken und je einem (entsprechenden) Chrominanzblock. Zu jedem Macroblock wird der günstigste "Partner" im früheren Bild gesucht, wobei nur die Luminanzblöcke zur Differenzbildung herangezogen werden. Der Bewegungsvektor der entsprechenden Chrominanzblöcke wird durch Halbierung des für die Luminanz berechneten gewonnen. Der Suchumfang wird auf ± 15 Pixel in vertikaler und horizontaler Richtung für jeden Macroblock eingeschränkt.

Eine Besonderheit des Standards ist in den Tab. 32, 33, 34 dargestellt: der Loopfilter. Er ist ein Tiefpaßfilter, der für "innere Blockpunkte" (im Beispiel sind dies bei horizontaler Anwendung die beiden mittleren Spalten und bei vertikaler die beiden mittleren Zeilen) den "neuen" Pixelwert durch die

110	218	116	112
108	210	110	114
110	218	210	112
112	108	110	116

Tabelle 32: Ein 4×4 Beispielblock...

110	165	140	112
108	159	135	114
110	188	187	112
112	109	111	116

Tabelle 33: ...wird zuerst horizontal...

Formel

$$x_{neu} = x/2 + x'/4 + x''/4$$

berechnet, wobei x' bzw. x'' die Pixelwerte der beiden Nachbarn sind. So ergibt sich der Wert in der ersten Zeile, zweite Spalte in Tab. 33 durch $165 = 110/4 + 218/2 + 116/4$ (ganzzahlige Division!), und der Wert in der zweiten Zeile, erste Spalte in Tab. 34 durch $108 = 110/4 + 108/2 + 110/4$ (beachte, daß bei "normaler Division" sich hier 109 ergäbe). Wie in Abb. 35 zu sehen, ist solch ein Loopfilter in die Macroblocksuchschleife eingebaut; genauer gesagt: paßt ein solchermaßen tiefpaßgefilterter Luminanzanteil des Macrovergleichsblocks "am besten" bei der "Partnersuche", so wird dies dem Decodierer mit einem Extra-Bit (Loopfilter-Status) mitgeteilt.

Wie bei JPEG ist die im ITU-T H.261 Standard vorgesehene Transformation die DCT, angewendet jeweils auf 8×8-Blöcke. Da nun aber teilweise auf Pixeln [sog. "intra mode"] und auf Pixeldifferenzen [sog. "inter mode"] (Bewegungskompensation!) gearbeitet wird, haben die Koeffizienten sehr unterschiedliche Charakteristika, was die einheitliche Quantisierung erschwert. H.261 beantwortet diese Schwierigkeit, indem zwischen zweiunddreißig verschiedenen Gleichquantisierern für die AC-Koeffizienten von Macroblock zu

110	165	140	112
108	167	148	113
110	161	154	113
112	109	111	116

Tabelle 34: ...und dann vertikal gefiltert.

Macroblock hin- und hergeschaltet werden kann. Da die Übertragung der quantisierten DCT-Koeffizienten wie bei JPEG nach Zickzackabtastung geschieht, ist bei den AC-Koeffizienten Null stets auch ein Repräsentant im Quantisierer. Es ist zu beachten, daß große Schrittweiten zwar evtl. eine schlechte Güte, andererseits aber einen ziemlich großen "Schwanz" von Nullen mit sich bringen, die Quantisiererwahl also auch die Übertragungsrate beeinflußt. Um der Notwendigkeit zu begegnen, für jeden Macroblock einen eigenen Quantisierschlüssel zu übertragen, werden 3×11 Macroblöcke zu einer "Blockgruppe" (GOB=group of blocks) organisiert, und für jede GOB wird ein Default-Quantisierer im Kopf übertragen.

Bei Videokonferenzen steht den Teilnehmern nur eine beschränkte Übertragungsrate zur Verfügung. Andererseits ist –im wesentlichen bedingt durch unterschiedlich "günstige" zu codierende Lauflängen und Nullschwänze nach der Quantisierung der zickzackabgetasteten DCT-Koeffizienten– je Block eine unterschiedliche Anzahl Bits zu übertragen. Deshalb werden die Daten vor dem Versenden zunächst kurz zwischengepuffert (dies ist nicht in Abb. 35 dargestellt). Da diese Pufferung unerwünschte Übertragungsverzögerungen mit sich bringt, ist der Puffer nur recht klein. Droht er überzulaufen, so wird eine entsprechende Nachricht an den "Quantisierer" gesendet, doch bitte die Rate zu senken, sei es durch Wahl eines Quantisierers mit größerer Schrittweite oder (im schlimmsten Fall) durch Auslassen eines Blocks bei der Übertragung.

10.1.4 MPEG

Die "Moving Pictures Expert Group" (MPEG) hat eine Reihe von Vorschlägen zur Bewegtbildcodierung gemacht, die sich vor allem hinsichtlich ihrer Funktionalität unterscheiden. So soll MPEG-1 für Übertragungsraten um 1,5 MBits/sec. gute Ergebnisse liefern, MPEG-2 für 10 MBits/sec. und MPEG-3 für 40 MBits/sec. Da MPEG-2 auch schon die für MPEG-3 gemachten Forderungen erfüllte, werden wir im folgenden MPEG-1, MPEG-2 und (kurz) MPEG-3 diskutieren.

MPEG-1 Das Grundschema von Abb. 35 ist auch hier gültig. Wesentlicher Unterschied zu H.261 ist die primäre Ausrichtung der MPEG-Standards auf Multimedia-Anwendungen, was insbesondere bedeutet, daß man eine Art "wahlfreien Zugriff" auf die Bildfolge gewährleisten muß; schon beim unidirektionalen (digitalen) Fernsehen will der Zuschauer ja evtl. sich später in einen laufenden Film einschalten, was aber unmöglich ist, wenn über die Bewegungskompensation immer auf beliebig weit zurückliegend gesendetete Information zugegriffen wird. Daher werden nicht zeitprädikativ codierte

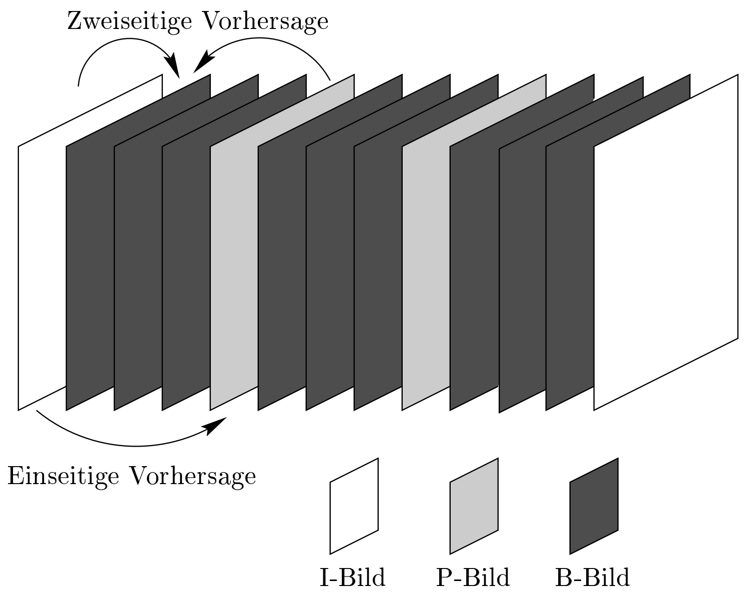


Abbildung 36: Eine mögliche Bildgruppe

Bilder periodisch übertragen (sog. I-Bilder). Daneben gibt es sog. P-Bilder ("prediction coded"), die im wesentlichen dem H.261-Standard entsprechen, nur daß –aus gleich verständlich werdenden Gründen– P-Bilder (die mit den I-Bildern zusammen auch Ankerbilder genannt werden) nicht unbedingt auf das letzte Bild der zu sendenden Bildfolge, sondern beispielsweise auf das fünfletzte zurückgreifen; dies ist natürlich auch bei der Bewegungskompensation zu berücksichtigen (Es wird empfohlen, den Suchraum für Macroblöcke in Abhängigkeit vom "Abstand" gesendeter Ankerbilder anzupassen). Um eine gute Komprimierung zu gewährleisten, gibt es in MPEG zusätzliche B-Bilder ("bidirectionally prediction coded"), die Korrelationen sowohl zum in der zu sendenden Bildfolge vorausgehenden Ankerbild als auch zum in der zu sendenden Bildfolge später kommenden Ankerbildes ausnutzen, was bei Szenenwechseln sinnvoll ist. Eine "Bildgruppe" (GOP=group of pictures) enthält mindestens ein I-Bild und ist die kleinste Bildfolge, in die sich ein Benutzer "zuschalten" kann, s. Abb. 36.

Die GOP-Struktur von MPEG-1 ist auch vorteilhaft gegen eine mögliche Fehlerfortpflanzung; aus ähnlichen Gründen wirkt sich die rückgekoppelte Ratensteuerung nur auf die Übertragung der "unwichtigen" B-Bilder aus.

Wegen der Abhängigkeiten zwischen den verschiedenen Bildern unterscheidet man zwischen der "Anzeigeordnung" (display order), also der Reihenfolge, in der die Bilder zu senden sind, und der "Übertragungsordnung" (bitstream order), der Reihenfolge also, in der die Bilder übertragen wer-

I	B	B	P	B	B	P	B	B	P	B	B	I
1	2	3	4	5	6	7	8	9	10	11	12	13

Tabelle 35: Anzeigeordnung einer GOP

I	P	B	B	P	B	B	P	B	B	I	B	B
1	2	3	4	5	6	7	8	9	10	11	12	13

Tabelle 36: Übertragungsordnung derselben GOP

den. Wie im Beispiel Tab. 35, 36 zu erkennen ist, werden zunächst die Ankerbilder übertragen und dann die sich unmittelbar darauf beziehenden B-Bilder.

MPEG-1 liefert eine Qualität, die mit VHS-Videos vergleichbar ist, wenn nicht zuviel Bewegung darin enthalten ist.

MPEG-2 ist eigentlich eine Sammlung von Algorithmen(empfehlungen) bzw. Spezifikationen, jeweils angepaßt auf definierte sog. Profile (Anforderungen) und Levels. Es gibt deren fünf Profile; in der Reihenfolge aufsteigender Anforderungen heißen sie: simple, main, SNR-scalable, spatially scalable, high. Höhere Profile können jeweils niedere decodieren. Die vier Levels sind: low, main, high 1440 und high; sie definieren die Bildauflösung (352×240, 720×480, 1440×1152 und 1920×1080) und damit die zu übertragende Datenmenge. Übrigens benutzt digitales HDTV MPEG-2 mit "main" Profil und "high" Level.

Die Profile "simple" und "main" entsprechen im wesentlichen dem zu MPEG-1 Gesagten; "simple" verwendet keine B-Bilder. Die drei übrigen Profile benutzen mehr als einen Bitstrom zur Codierung. Der Grundbitstrom ist eine niederratige Codierung der Bildfolge; die anderen Bitströme dienen zur Bildqualitätserhöhung. Bei Netzwerkübertragung braucht bei Überlast nur der Grundbitstrom übertragen zu werden, um so die Bitrate zu senken. Daher werden diese Profile auch "skalierbar" genannt. Ideen ähnlich der von der fortschreitenden Bildübertragung kommen zur Anwendung, nur daß die Filter sowohl zeitlich als auch räumlich arbeiten, s. Abb. 37.

MPEG-2 gestattet –im Gegensatz zu MPEG-1– aufwendigere Bewegungskompensation, indem

- auch Informationen des schon übertragenen Teils des aktuellen Bildes benutzt werden und

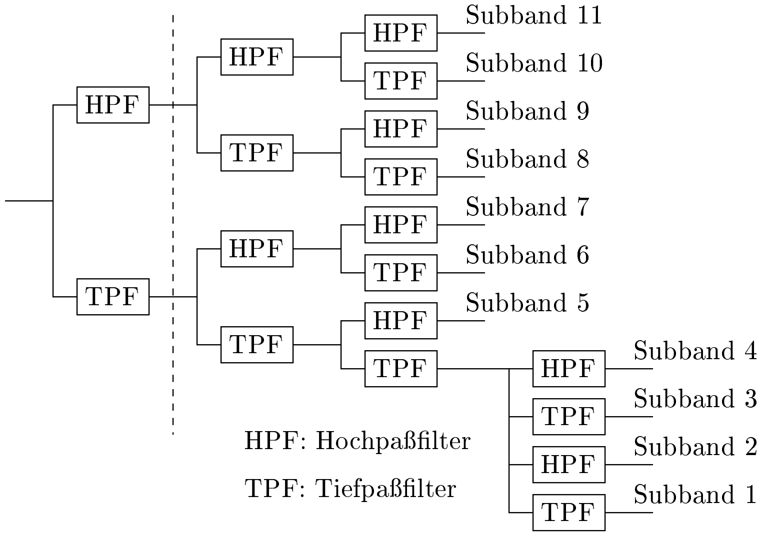


Abbildung 37: Filteranalyse nach Karlsson/Vetterli

- unterschiedliche Bewegungsrichtungen von 16×8-Sub-Macroblöcken berücksichtigt werden (können).

MPEG-4 ist eine Antwort auf (im wesentlichen auf die Zukunft gerichtete) Schlagwörter wie "interaktives Fernsehen", "multiperspektives Fernsehen", "Video on demand", etc.

Nach Rautenberg [8] lassen sich die Funktionalitäten wie in Tab. 37 darstellen. Es folgen kurze Erläuterungen:

1. Ein Bild wird aus verschiedenen, priorisierten audiovisuellen Objekten (AVOs) zusammengesetzt gedacht, die jeweils mit unterschiedlicher Genauigkeit übertragen werden können. Natürlich ist es schwierig, solche AVOs automatisch zu identifizieren; dem Hersteller von MPEG-4-Filmen ist es aber freigestellt, diese (aus seinem Wissen heraus) zu definieren.

2. Diese Funktionalität (die z.B. zu einem Hobby "Editieren von Filmen" führen könnte) soll dadurch gewährleistet werden, daß Bewegungsinformationen von AVOs durch (in einer Erweiterung von C++ geschriebenen) "Kompositionsskripte" übertragen werden.

3. Werkzeuge können sein: Indizes, Hyperlinks, ...

4. Hybride Codierung soll auch Multiperspektivität unterstützen.

Funktionalitäten	Kategorien
1. Inhaltsbasierte Skalierbarkeit 2. Inhaltsbasierte Interaktivität und Editieren von Datenströmen 3. Werkzeuge für den inhaltsbasierten Zugriff auf Multimedia Daten 4. Hybride Kodierung von Daten mit natürlichem und synthetischem Ursprung	I. Inhaltsbasierte Interaktivität
5. Kodierung mehrfacher, nebenläufiger, zeitlich verzahnter Datenströme 6. Verbesserte Kodierungsleitung	II. Kompression
7. Robustheit in fehlerträchtigen Umgebungen 8. Verbessertes Zeitverhalten bei wahlfreiem Zugriff	III. Universeller Zugriff

Tabelle 37: Funktionalitäten bei MPEG-4

5. Codierungen mehrfacher Datenströme können beispielsweise mehrere Tonspuren (Sprachen) derselben Szene oder mehrere Bildperspektiven (3D-Fernsehen) betreffen; die dabei erzeugte Redundanz sollte durch intelligente Komprimierung kompensiert werden.

8. Z.B. soll der “schnelle Suchlauf rückwärts” für ein selektiertes AVO möglich sein.

10.2 Audio-Daten

Natürlich sind bei Filmen beispielsweise nicht nur Bild- sondern auch Toninformationen zu übertragen. Für eine gute Kompression von solchen Audio-Daten (insbesondere Sprache) ist zu berücksichtigen:

1. eingeschränkter hörbarer Frequenzbereich;

2. besondere Empfindlichkeit für Tonhöhen;

3. Lautstärkewahrnehmung ist mit Tonhöhenwahrnehmung korreliert;

4. in europäischen Sprachen gibt es deutlich voneinander trennbare “stimmhafte” (sinusartige) und “stimmlose” (rauschartige) Laute; ein Decodierer für solche Sprachen kann daher wie in Bild 38 dargestellt arbeiten.

Wie kann der Codierer eine Entscheidung darüber treffen, ob ein Laut stimmhaft oder stimmlos ist? Eine Möglichkeit ist die Filteranalyse; denn Rauschsignale sind hochfrequenter als sinusartige.

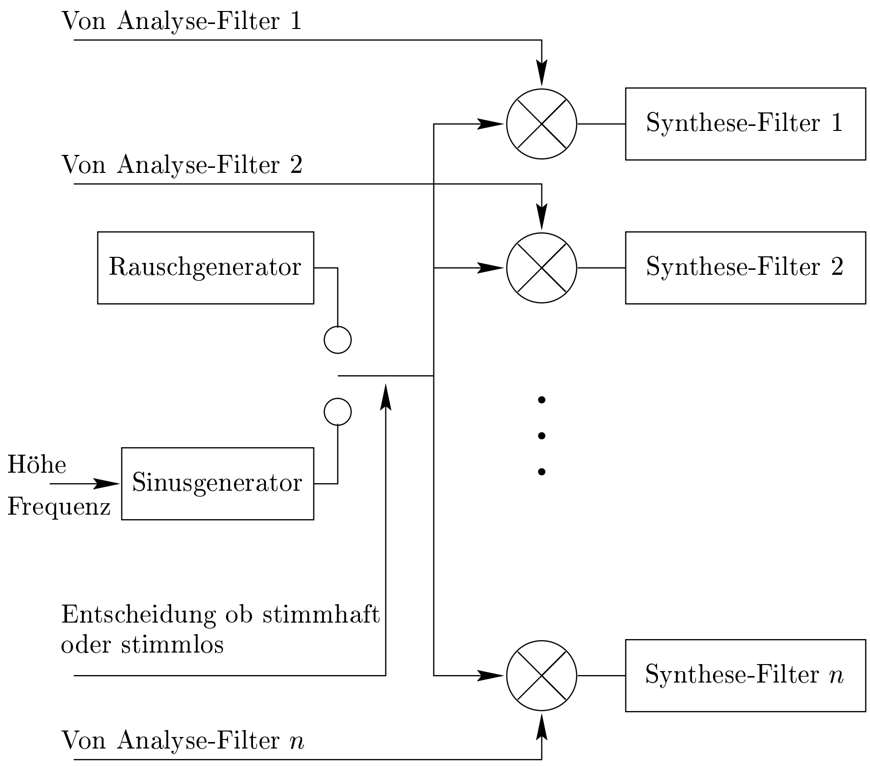


Abbildung 38: Schema eines Sprachdecodierers

Der Tonhöhenempfindlichkeit wird beispielsweise im auch bei MPEG verwendeten CELP ("code excited linear prediction" von Atal und Schroeder) durch eine spezielle adaptive Codebuchtechnik für die Erregungssignalformen Rechnung getragen.

Interessante Adressen aus dem Internet

Allgemeine Informationen

`www.internz.com/compression-pointers.html`
`www2.echo.lu/oii/en/compress.html`
`HTTP.CS.Berkeley.EDU/ guyb/real-world/compress/index.html`
`www.cis.ohio-state.edu/hypertext/faq/bngusenet/comp/compression/top.html`

Speziellere Informationen

JPEG:
`www.cs.sfu.ca/undergrad/CourseMaterials/CMPT479/`
`material/misc/interactive_jpeg/Ijpeg.html`

Wavelet-Verfahren:
`www.wavelet/org`
`www.iro.umontreal.ca/ pigeon`
`www.summus.com`

Fraktale Kompression:
`www.iterated.com`
`www.informatik.uni-leipzig.de/cgip/projects/fic.html`

Kommerzielle Kombination Wavelet/Fraktal:
`www.OLiVR.com` (Vorsicht: User-Registrierung)

Literatur

[1] K. Bosch. *Elementare Einführung in die angewandte Statistik,* Vieweg, 1987.

[2] R.J. Clarke. *Transform Coding of Images,* Academic Press, 1985 u. 1990.

[3] O. Egger, P. Fleury, T. Ebrahimi, M. Kunt. *High-Performance Compression of Visual Information—A Tutorial Review— Part I: Still Pictures,* Seiten 974–1011 in: Proc. IEEE, Band 87, Nr. 6, Juni 1999. **Hinweis:** Diese Zeitschrift enthält meist empfehlenswerte Übersichtsartikel zu verschiedendsten Gebieten der Informatik.

[4] G. Kaiser. *A Friendly Guide to Wavelets,* Birkhäuser, 1994.

[5] J. Loviscach, W. Fastenrath. *Villa Kunterbunt; Farbkorrektur mit Color-Management-Systemen,* Seiten 180 ff. in: c't, Magazin für Computertechnik, Nr. 10, Oktober 1996.

[6] J.-R. Ohm. *Digitale Bildcodierung,* Springer, 1995.

[7] S. Pigeon. *Image Compression with Wavelets,* in: Dr. Dobb's Journal, August 1999.

[8] M. Rautenberg. *MPEG-4: Ein neuer Standard für mehr Funktionalität in Multimedia-Anwendungen,* Seiten 82–87 in: Informatik-Spektrum, Band 22, Heft 2, April 1999.

[9] D. Salomon. *Data Compression; The Complete Reference,* Springer, 1998.

[10] K. Sayood. *Introduction to Data Compression,* Morgan Kaufmann Publishers, Inc., 1996.

[11] J. M. Shapiro. *Embedded Image Coding Using Zerotrees of Wavelet Coefficients,* Seiten 3445–3462 in: IEEE Transactions on Signal Processing, Band 51, Nr. 12, Dezember 1993. **Hinweis:** Diese Zeitschrift enthält sehr häufig Artikel zum Thema Kompressionsverfahren.

[12] U. Simon, M. Berndtgen. *Handlich bunt; Kompressionstechniken für Bild- und Videodateien im Vergleich,* Seiten 222 ff. in: c't, Magazin für Computertechnik, Nr. 11, November 1996.

[13] F. Topsøe. *Informationstheorie,* Teubner, 1974.

[14] M. Zeller. *Flinkes Wellenspiel; Signalverarbeitung mit Wavelets,* Seiten 258 ff. in: c't, Magazin für Computertechnik, Nr. 11, November 1994.