



3 Kompression

3.1 Einführung

Motivation - Datenvolumen



- Text 1 Seite mit 80 Zeichen/Zeile,
64 Zeilen/Seite, 1 Byte/Zeichen =
 $80 * 64 * 1 * 8 = 40 \text{ kBit/Seite}$
- Standbild 24 Bit/Pixel, $512 * 512$ Pixel/Bild =
 $512 * 512 * 24 = 8 \text{ MBit/Bild}$
- Audio CD-Qualität, Abtastrate 44,1 KHz,
16 Bit pro Abtastwert =
 $44,1 * 16 = 706 \text{ kBit/s}$
Stereo: 1,412 MBit/s
- Video Vollbild mit $1024 * 1024$ Pixel/Bild,
24 Bit/Pixel, 30 Bilder/s =
 $1024 * 1024 * 24 * 30 =$
720 MBit/s



Motivation



- Sowohl zur Speicherung als auch zur Übertragung ist eine Datenkompression erforderlich



Kompression



- Fast alle Daten enthalten Redundanz
 - „eins, zwei, drei“ vs. „1, 2, 3“
- Unterdrückung von Redundanz erhöht die Informationsrate
 - Geringerer Speicherungsbedarf
 - Weniger Übertragungskapazität



Kompression



- Bei manchen Medien ist die Einführung von nicht wahrnehmbarem Verlust möglich
- Probleme
 - Rechenaufwand
 - Fehleranfälligkeit
 - Bitfehler, Paketverlust, etc.
 - Eventuell Informationsverlust



Kompressionsarten



- Verlustfreie Kompression (lossless compression)
 - Das Original kann vollständig wiedergewonnen werden
 - Typische Kompressionsraten von 2:1 bis ca. 50:1



Kompressionsarten



- Verlustbehaftete Kompression (lossy compression)
 - Unterschied zwischen Originalobjekt und dekodiertem Objekt
 - Physiologische und wahrnehmungspsychologische Eigenschaften des Auges und des Ohres werden ausgenutzt
 - Höhere Kompressionsraten als bei verlustfreier Kompression möglich (50:1 bis über 100:1)



Kompressionsarten



- Entropie-Kodierung
 - Korrelationen finden und ausnutzen
 - Nullunterdrückung, Lauflängen-Kodierung
 - Beispiele
 - Huffman Coding
 - Lempel-Ziv (LZW, etc.)
 - Verlustlos
 - Weiterverarbeitung möglich



Kompressionsarten



- Differenzen-Kodierung
 - in Raum und Zeit
 - $d_i = |val_i - val_{i+1}| < \epsilon$
 - Delta-Modulation
 - Vorhersage



Kompressionsarten



- Truncation Coding
 - Grenzen der menschlichen Wahrnehmung ausnutzen
 - z.B. Runden von Gleitpunkt-Zahlen
 - Insbesondere bei Audio, Video
 - Verlustbehaftet
- Transformationskodierung
 - Daten aus einem anderen Blickwinkel betrachten
 - Koordinatentransformation
 - Insbesondere für Bilder



Kompressionsarten



- Modellbasierte Kodierung
 - Decoder = eine Maschine die einen Medienstrom erzeugt
 - Coder findet die richtigen Eingaben für diese Maschine
 - Verwendet für Audio, Video mit sehr niedriger Bitrate





3 Kompression

3.2 Entropie-Kodierung

Lauf­längen­kodierung



- run-length encoding
- Prinzip
 - Ersetze Wiederholungen desselben Zeichens im Text („run“) jeweils durch einen Zähler und das Zeichen



Lauf­längen­kodierung



- Beispiel
 - Text:
AAAABBBBAABBBBBCCCCCCCCDABCBAABBBB
 - Kodierung:
4A3B2A5B8C1D1A1B1C1B2A4B
- Es ist nur dann eine gute Kompressionsrate zu erwarten, wenn es häufig lange „runs“ gibt
 - Folgen von Leerzeichen
 - Führende Nullen
 - identische Farben in Bildern



Lauf­längen­kodierung für Binärdateien



- Bei Binärdateien folgt jedem Run von Einsen immer ein Run von Nullen und umgekehrt
- Es genügt deshalb, nur die Länge der Runs zu speichern



Laufängenkodierung für Binärdateien



- Beispiel

000000000000111111111111111100000000	12	14	9		
00000000000111111111111111111111000000	10	18	7		
0000000111111111111111111111111110000	7	24	4		
0000001111111111111111111111111111000	6	26	3		
0000111111111111111111111111111111110	4	30	1		
00011111110000000000000000000000111111	3	7	18	7	
000111110000000000000000000000000011111	3	5	22	5	
00011100000000000000000000000000000111	3	3	26	3	
00011100000000000000000000000000000111	3	3	26	3	
00011100000000000000000000000000000111	3	3	26	3	
00011100000000000000000000000000000111	3	3	26	3	
000011110000000000000000000000000001110	4	4	23	3	1
0000001110000000000000000000000000111000	4	3	20	3	3
000000011111111111111111111111111110000	7	24	4		



Kodierung mit variabler Länge



- klassische Zeichencodes verwenden gleich viele Bits für jedes Zeichen
- wenn verschiedene Zeichen mit unterschiedlichen Häufigkeiten vorkommen
 - Verwende für häufige Zeichen wenige Bits
 - und für seltene Zeichen mehr Bits



Kodierung mit variabler Länge



- Code 1: **A B C D E ...**
 1 2 3 4 5 ... (binär)
 - Kodierung von ABRACADABRA mit konstanter Zeichengröße (5 Bit):
0000100010100100000100011000010010000
001000101001000001
- Code 2: **A B R C D**
 0 1 01 10 11
 - kodierter Text:
0 1 01 0 10 0 11 0 1 01 0



Kodierung ohne explizite Begrenzer



- Code 2 kann nur eindeutig dekodiert werden, wenn Zeichenbegrenzer mitgespeichert werden
 - vergrößert die Datenmenge erheblich
- Idee
 - Keine Kodierung eines Zeichens darf mit dem Anfang der Kodierung eines anderen Zeichens übereinstimmen
 - D.h. kein Codewort ist Präfix eines anderen Codeworts
 - Dann kann auf Begrenzer verzichtet werden



Kodierung ohne explizite Begrenzer



- Code 3:

A	11
B	00
R	011
C	010
D	10

- kodierter Text:

1100011110101110110001111



Darstellung als Trie



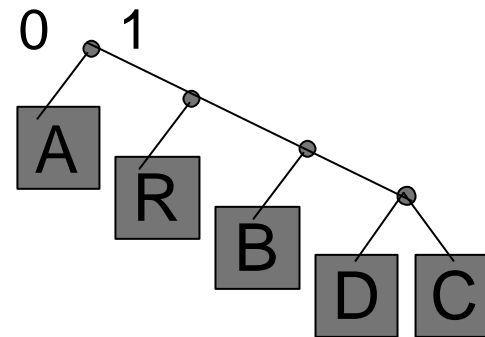
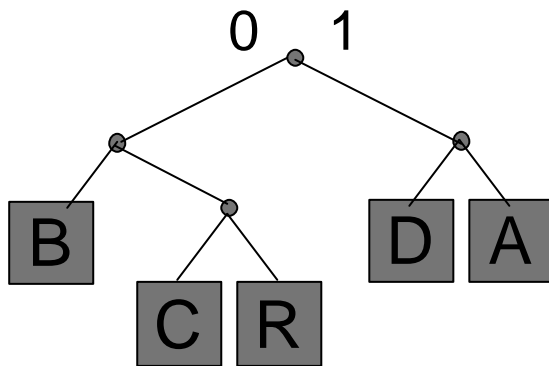
- jeder beliebige Trie mit M äußeren Knoten kann jede beliebige Zeichenfolge mit M verschiedenen Zeichen kodieren
- Der Code für jedes Zeichen wird durch den Pfad von der Wurzel zum Zeichen bestimmt
 - mit 0 für „nach links gehen“ und 1 für „nach rechts gehen“



Darstellung als Trie



- Die Triedarstellung garantiert, die Präfixfreiheit
- die Zeichenfolge lässt sich eindeutig dekodieren



Huffman Code



- Frage
 - wie findet man für eine Kodierung mit variabler Länge für gegebene Zeichenwahrscheinlichkeiten die optimale Bitkodierung ?



Algorithmus von D.Huffman (1952)



1. Bestimme die Auftretshäufigkeiten der Zeichen und schreibe sie an die Blattknoten eines aufzubauenden Binärbaums
2. Nimm die bisher unerledigten zwei Knoten mit den geringsten Häufigkeiten und berechne deren Summe
3. Erzeuge einen Elternknoten für diese beiden und beschrifte ihn mit der Summe. Markiere die Verzweigung nach links mit 0, nach rechts mit 1
4. Markiere die beiden bearbeiteten Knoten als erledigt. Wenn es nur noch einen nicht erledigten Knoten gibt, terminiere. Sonst weiter mit 2.

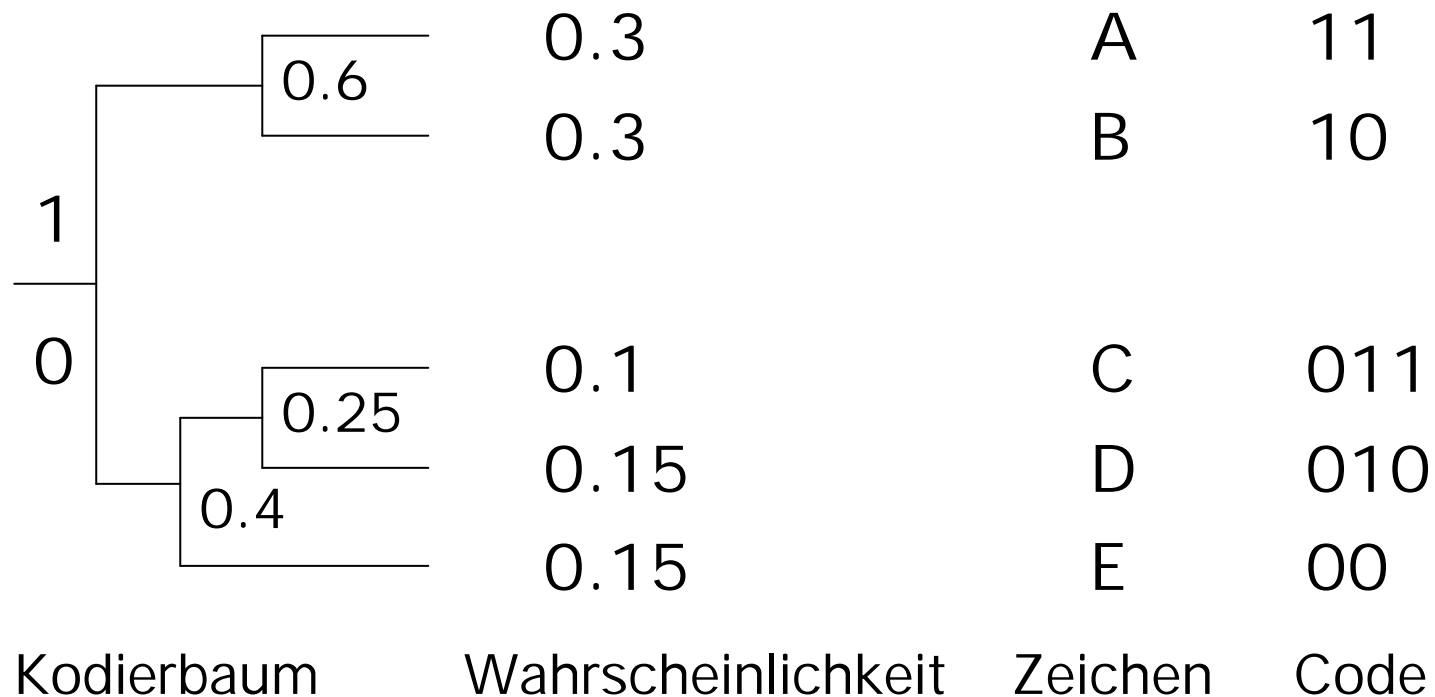


Beispiel Huffman Code



- Wahrscheinlichkeiten der Zeichen:

- $p(A) = 0.3$; $p(B) = 0.3$; $p(C) = 0.1$;
 $p(D) = 0.15$; $p(E) = 0.15$



Optimalität des Huffman - Code



- Zeichen mit großen Häufigkeiten sind näher an der Wurzel des Baumes
 - Sie haben kürzere Codewortlänge
- Dies ergibt einen optimalen Code



Optimalität des Huffman - Code



- Die Länge einer kodierten Zeichenfolge ist gleich der gewichteten äußeren Pfadlänge des Huffman-Baumes
- Die gewichtete äußere Pfadlänge eines Baumes ist gleich der über alle äußeren Knoten gebildeten Summe der Produkte des Gewichts mit der Entfernung von der Wurzel
- Kein Baum mit den gleichen Häufigkeiten bei den äußeren Knoten hat eine kleinere gewichtete äußere Pfadlänge als der Huffman-Baum



Beweisidee



- Mit Hilfe des gleichen Prozesses kann ein beliebiger anderer Binärbaum konstruiert werden, doch ohne bei jedem Schritt unbedingt die zwei Knoten mit dem kleinsten Gewicht auszuwählen
- Mittels Induktion lässt sich beweisen, dass keine Strategie zu einem besseren Ergebnis führen kann als die, bei der jeweils zuerst die beiden kleinsten Gewichte ausgewählt werden



Dekodierung von Huffman Codes



- Dekodierung unter Verwendung des Tries
 1. Lies den Eingabestrom sequenziell und traversiere den Trie, bis ein Blattknoten erreicht ist
 2. Gib bei Erreichen des Blattknotens das erkannte Zeichen aus
- Eigenschaften
 - die Eingabe-Bitrate ist konstant
 - die Ausgabe-Zeichenrate ist variabel



Dekodierung von Huffman Codes



- Verwendung einer Dekodiertabelle
- Erzeugung der Tabelle
 - Hat das längste Codewort L Bits, dann hat die Tabelle 2^L Einträge
 - Sei c_i das Codewort für Zeichen s_i . c_i habe l_i Bits. Wir erzeugen 2^{L-l_i} Einträge in der Tabelle, bei denen jeweils die ersten l_i Bits = c_i sind und die verbleibenden $L-l_i$ Bits alle möglichen Kombinationen annehmen
 - An all diesen Adressen wird s_i als erkanntes Zeichen eingetragen und l_i als Codewortlänge vermerkt



Dekodierung von Huffman Codes



- Einsatz der Tabelle zur Dekodierung
 1. Lies L Bits aus dem Eingabestrom in einen Puffer
 2. Benutze den Puffer als Adresse in die Tabelle und gib das erkannte Zeichen s_i aus
 3. Entferne die ersten l_i Bits aus dem Puffer und ziehe weitere l_i Bits aus dem Eingabestrom nach
 4. Weiter mit Schritt 2
- Eigenschaften
 - Tabellenverfahren sind schnell
 - die Ausgabe-Zeichenrate ist konstant
 - die Eingabe-Bitrate ist variabel



Bewertung des Huffman Code



- sehr guter Code für viele praktische Zwecke
- nur geeignet, wenn die Häufigkeiten der Zeichen a priori bekannt und immer gleich oder ähnlich sind
- Variante
 - Ermittle die Häufigkeiten für jeden gegebenen Text neu und speichere/übertrage den Code mit den Daten



Bewertung des Huffman Code



- Ein kleiner „Verlust“ entsteht, da jedes Zeichen mit einer ganzzahligen Anzahl Bits kodiert werden muss
 - die Codelänge kann den Häufigkeiten nicht theoretisch optimal angepasst werden
- Verbesserung
 - arithmetische Kodierung (kommt später)



Wörterbuch-basierte Kodierung



- Wörterbuch
 - Tabelle von Zeichenketten, auf die bei der Kodierung Bezug genommen wird
- Beispiel
 - „Vorlesung“ stehe im Wörterbuch auf Seite x_3 , Zeile y_3
 - Referenzierung durch (x_3, y_3)
 - „Heute ist Vorlesung“ wird dann z.B. kodiert als Sequenz von Referenzen
 $(x_1, y_1) (x_2, y_2) (x_3, y_3)$



Wörterbuch-basierte Kodierung



- Statische Verfahren
 - Das Wörterbuch wird vor der Kodierung festgelegt und nicht mehr geändert



Wörterbuch-basierte Kodierung



- Dynamische Verfahren
 - Das Wörterbuch wird basierend auf der zu übertragenden Nachricht beim Sender bzw. Empfänger aufgebaut
 - Wichtige Verfahren beruhen auf zwei Algorithmen von Abraham Lempel und Jacob Ziv (1977/78)
 - LZ77 gzip, PKZIP, PNG
 - LZW, Lempel-Ziv-Welch (1984)
GIF, Unix compress (Unisys Patent)



LZW Kodierung



- Startet mit einem Wörterbuch, das die Einzelzeichen enthält
- Kodierungs-Algorithmus

```
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
```



LZW Beispiel



- Eingabe: abababcabac

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



LZW Dekodierung



- Startet mit dem Wörterbuch der Einzelzeichen
- Algorithmus:

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



LZW Eigenschaften



- die Code-Tabelle (das Wörterbuch) muss nicht übertragen werden
- In der Praxis wird die Länge der Tabelle begrenzt
 - Trade-Off zwischen Geschwindigkeit und Kompressionsrate
- LZW passt sich dynamisch an die Eigenschaften der Zeichenkette an



LZW Eigenschaften



- Komplexität
 - Kodierung: $O(N)$,
N = Länge der nichtkodierten Nachricht
 - Dekodierung: $O(M)$,
M = Länge der kodierten Nachricht
 - Da $M \leq N$ ist die Dekodierung sehr effizient
- Typische Beispiele für Dateigrößen im Vergleich zum Original

Art	Huffman	Lempel-Ziv
C-Dateien	65 %	45 %
Maschinenkode	80 %	55 %
Text	50 %	30 %



Arithmetische Kodierung



- Informationstheoretisch gesehen ist die Huffman-Kodierung nicht ganz optimal
 - einem Datenwort wird immer eine ganzzahlige Anzahl Bits als Codewort zugewiesen
- Idee der arithmetischen Kodierung
 - Eine Nachricht wird als Gleitkommazahl aus dem Intervall $[0, 1)$ kodiert
 - das Intervall wird nach der Wahrscheinlichkeit der einzelnen Symbole aufgeteilt
 - Jedes Intervall repräsentiert ein Zeichen



Kodierungs - Algorithmus



1. Zerlege das Intervall $[0,1)$ in je ein Teilintervall für jedes Zeichen des Alphabets entsprechend der Zeichenwahrscheinlichkeiten.
2. Das nächste zu kodierende Zeichen bestimmt das nächste Basisintervall. Teile dieses Intervall wieder nach den Zeichenwahrscheinlichkeiten. Führe dies so lange fort, bis ein Endesymbol erreicht wird oder eine bestimmte Anzahl Zeichen kodiert ist.
3. Wähle eine Zahl aus dem zuletzt entstandenen Intervall wird als Repräsentant für das Intervall derart, dass diese sich mit möglichst wenigen Bits darstellen lässt.



Dekodierungs - Algorithmus



1. Das Intervall $[0,1)$ wird wie beim Kodierungsprozess unterteilt
2. Der Code bestimmt das nächste zu unterteilende Basisintervall nach seinem Wert. Dieses Intervall steht für ein bestimmtes Zeichen, das damit dekodiert ist.
3. Führe dies so lange fort, bis das Endesymbol oder eine bestimmte festgelegte Anzahl von Zeichen dekodiert worden ist

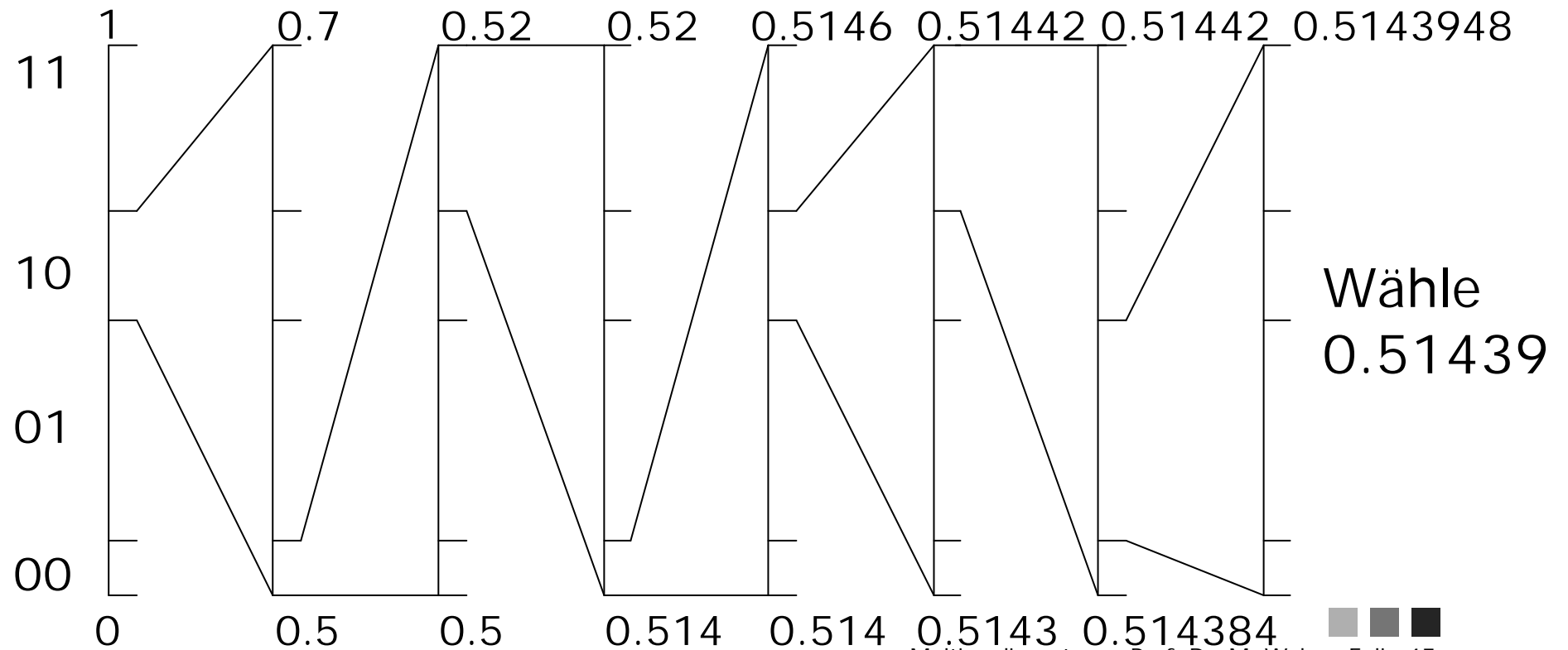


Statische arithmetische Kodierung - Beispiel



Zeichen, Alphabet	00	01	10	11
Wahrscheinlichkeiten	0.1	0.4	0.2	0.3
Erste Kodierungsintervalle	$[0,0.1)$	$[0.1,0.5)$	$[0.5,0.7)$	$[0.7,1)$

Nachricht: 10 00 11 00 10 11 01



Dekodierung des Beispiels



Schritt	Intervall	Zeichen	Dekodierungsentscheidung
1	$[0.5, 0.7)$	10	0.51439 befindet sich in $[0.5, 0.7)$
2	$[0.5, 0.52)$	00	0.51439 befindet sich im 1. Zehntel des Intervalls $[0.5, 0.7)$
3	$[0.514, 0.52)$	11	0.51439 befindet sich im 7. Zehntel des Intervalls $[0.5, 0.52)$
4	$[0.514, 0.5146)$	00	0.51439 befindet sich im 1. Zehntel des Intervalls $[0.514, 0.52)$
5	$[0.5143, 0.51442)$	10	0.51439 befindet sich im 5. Zehntel des Intervalls $[0.514, 0.5146)$
6	$[0.514384, 0.51442)$	11	0.51439 befindet sich im 7. Zehntel des Intervalls $[0.5143, 0.51442)$
7	$[0.51439, 0.5143948)$	01	0.51439 befindet sich im 4. Zehntel des Intervalls $[0.514384, 0.51442)$



Dynamische arithmetische Kodierung



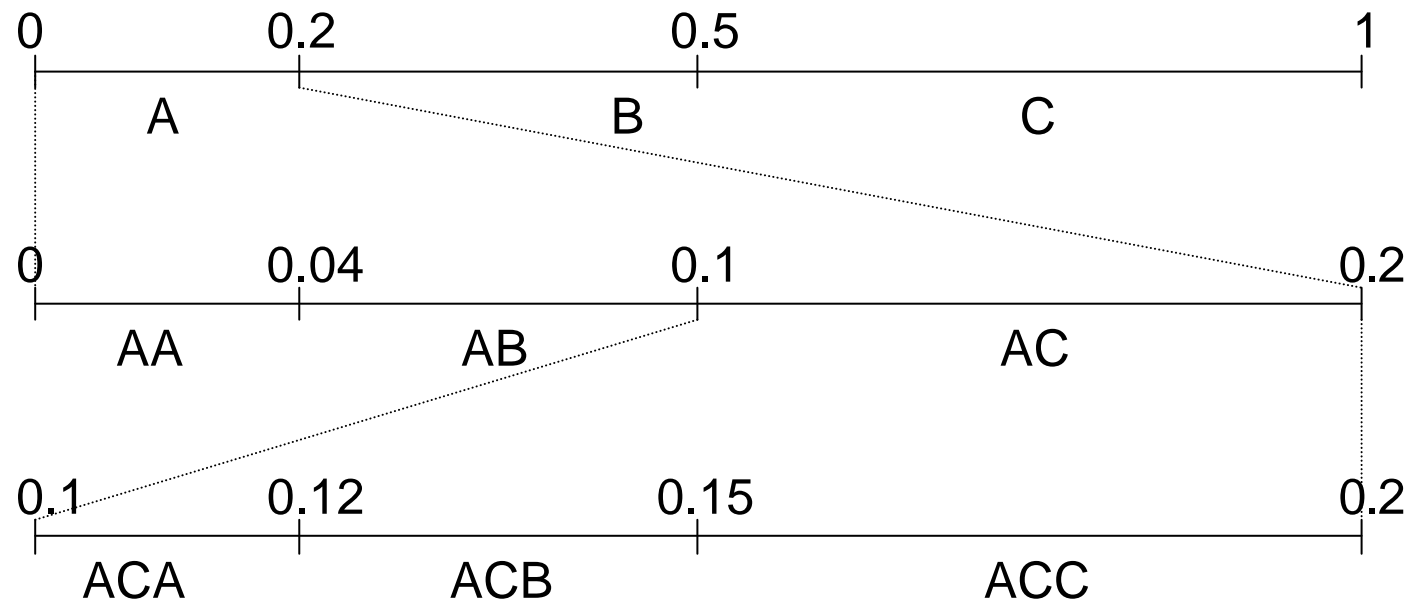
- Idee
 - Übertragung in Blöcken
 - Neuberechnung der Wahrscheinlichkeiten nach jedem Block
- Beispiel
 - Alphabet = {A,B,C}
 - Anfangswahrscheinlichkeiten = {0,2; 0,3; 0,5}
 - Nachricht: ACB AAB (in 3er-Blöcken)



Beispiel dynamische arithm. Kodierung



- 1. Block



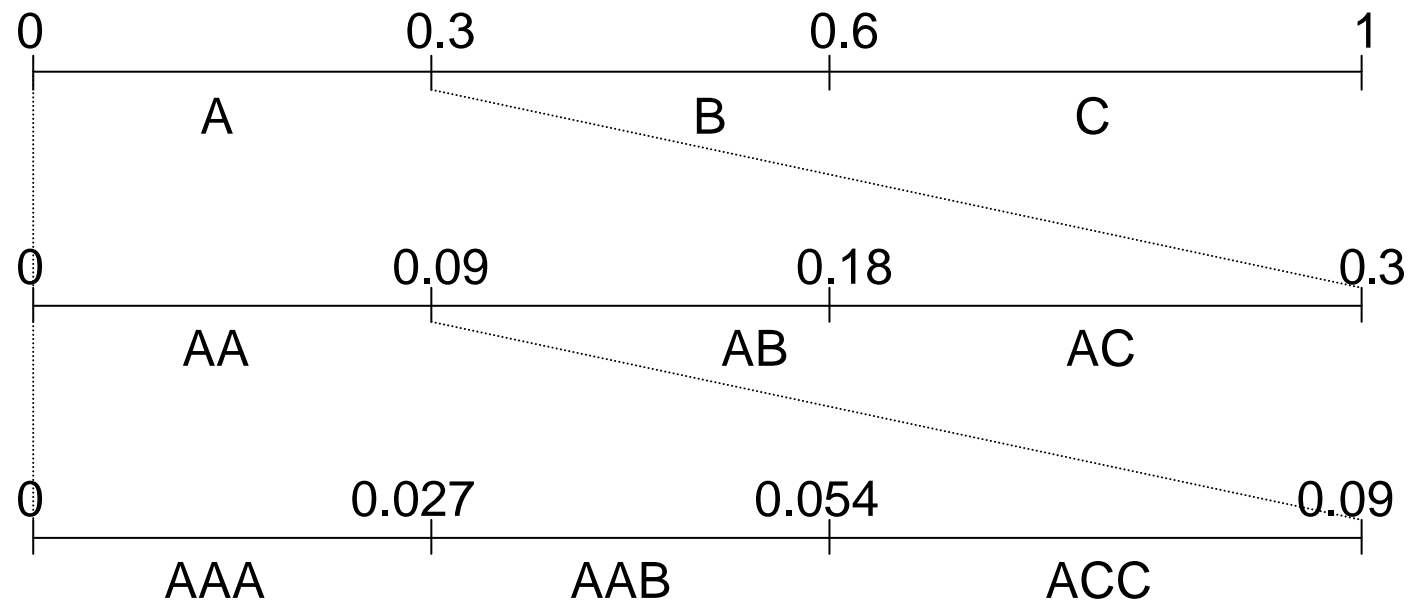
- Übertragung [0.12, 0.15), z.B. 0.12



Beispiel Fortsetzung



- Aktualisierung der Wahrscheinlichkeiten
 - $P_{\text{neu}} = (P_{\text{alt}} + P_{\text{aktuell}}) / 2$
 - Neue Wahrscheinlichkeiten = $\{0,3; 0,3; 0,4\}$
- Kodierung des zweiten Blocks



- Übertragung $[0.027, 0.054)$, z.B. 0.05



Eigenschaften der arithm. Kodierung



- Die Kodierung hängt von der Zeichenwahrscheinlichkeit und damit vom Kontext ab
- Die Wahrscheinlichkeit der Symbole bestimmt die Kompressionseffizienz
 - wahrscheinlichere Symbole werden in größeren Intervallen kodiert und brauchen damit weniger Bits



Eigenschaften der arithm. Kodierung



- Die Code-Länge ist theoretisch optimal
 - die Anzahl Bits pro Zeichen kann nicht-ganzzahlig sein und der Zeichenwahrscheinlichkeit genau angepasst werden
- Die Terminierung des Verfahrens beim Dekodieren ist auf mehrere Arten möglich
 - Die Nachrichtenlänge ist Sender und Empfänger bekannt
 - Es gibt nur eine bestimmte Anzahl von Nachkommastellen
 - auch dies müssen Sender und Empfänger wissen



Probleme der arithm. Kodierung



- Präzision von Maschinen ungenau
 - „overflow“ oder „underflow“ kann vorkommen
- Die Dekodierung ist erst nach vollständigem Erhalt der (Teil-)Nachricht möglich
 - Diese kann aus vielen Nachkommastellen bestehen
- Sehr fehleranfällig
 - ein Bitfehler zerstört die ganze (Teil-)Nachricht
- Exakte Wahrscheinlichkeiten sind oft nicht erhältlich
 - die maximale theoretische Code-Effizienz kann praktisch kaum erreicht werden





3 Kompression

3.3 Truncation Coding

Block Truncation Coding (BTC)



- einfaches Verfahren für Graustufen-Bilder
- Annahme
 - jedes Pixel im Original ist mit einem Grauwert [0..255] codiert
- Algorithmus
 1. Zerlege das Bild in Blöcke von $n \times m$ Pixel
 2. Berechne Mittelwert und Standardabweichung der Pixelwerte für jeden Block

$$\mathbf{m} = \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m Y_{i,j} \quad \mathbf{s} = \sqrt{\frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m (Y_{i,j} - \mathbf{m})^2}$$

mit $Y_{i,j}$ = Graustufe des Pixels



Block Truncation Coding (BTC)



- 3. Weise dem Block eine Bitmatrix der Größe $n \times m$ nach folgender Regel zu

$$B_{i,j} = \begin{cases} 1 & \text{falls } Y_{i,j} > m \\ 0 & \text{sonst} \end{cases}$$

- 4. Berechne zwei Graustufenwerte für den Block
 - a für den Wert der dunkleren Pixel, b für den Wert der helleren Pixel

$$a = m - s \sqrt{\frac{p}{q}} \quad b = m + s \sqrt{\frac{q}{p}}$$

- mit p = Anzahl der Pixel, heller als der Mittelwert
 - q = Anzahl der dunkleren Pixel
- 5. gebe (Bitmatrix, a, b) für jeden Block aus



BTC Beispiel



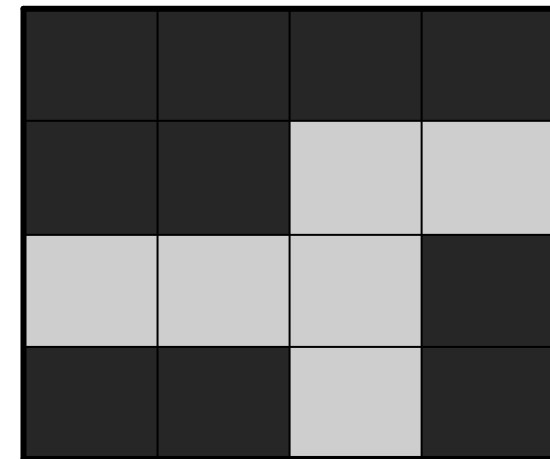
12	14	15	23
62	100	201	204
190	195	240	41
20	48	206	45

$\mu = 101$
→

0	0	0	0
0	0	1	1
1	1	1	0
0	0	1	0



$a = 38,$
 $b = 206$



BTC Kompressionsrate



- Beispiel
 - Blockgröße: 4 x 4
 - Original: 1 Byte pro Pixel
 - Kodierte Darstellung:
Bitmatrix mit 16 Bits +
2 x 8 Bits für a und b
- Reduktion von 16 Byte auf 4 Byte
- Kompressionsfaktor 4



BTC Beispiel



Color Cell Compression



- Verfahren zur Kompression von Farbbildern
- prinzipiell könnte man BTC separat für die Farbkanäle RGB anwenden
- CCC arbeitet jedoch auf dem YUV-Modell
 - qualitativ bessere Ergebnisse, da die Luminanz direkt lesbar ist



CCC Algorithmus



1. Zerlege das Bild in Blöcke der Größe $m \times n$ Pixels
2. Berechne für jedes Farbpixel die Helligkeit
 - $Y = 0.3 P_{\text{red}} + 0.59 P_{\text{green}} + 0.11 P_{\text{blue}}$
 $Y = 0$ entspricht schwarz, $Y = 1$ entspricht weiß
3. Für $c = \text{red, green, blue}$ berechne die mittleren Farbwerte der Pixel

$$a_c = \frac{1}{q} \sum_{Y_{i,j} \leq m} P_{c,i,j} \quad b_c = \frac{1}{p} \sum_{Y_{i,j} > m} P_{c,i,j}$$

- p ist die Anzahl der Pixel, die heller als der Mittelwert sind
- q die Anzahl der dunkleren Pixel



CCC Algorithmus



- 4. Weise dem Block eine Bitmatrix der Größe $n \times m$ nach folgender Regel zu

$$B_{i,j} = \begin{cases} 1 & \text{falls } Y_{i,j} \leq m \\ 0 & \text{sonst} \end{cases}$$

- 5. Die Werte $a = (a_{\text{red}}, a_{\text{green}}, a_{\text{blue}})$ und $b = (b_{\text{red}}, b_{\text{green}}, b_{\text{blue}})$ werden gemäß ihrer Nähe in eine Farbtabelle abgebildet. Es ergeben sich Werte a' und b' als Indizes der Farbtabelle
- 6. Ausgabe: (Bitmatrix, a' , b') für jeden Block



CCC Dekompression



- Dekompression für jeden Block durch Nachschauen in der Color Lookup Table

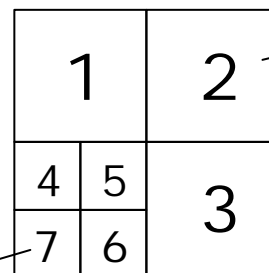
$$P'_{i,j} = \begin{cases} CLUT[a'] & \text{falls } B_{i,j} = 1 \\ CLUT[b'] & \text{sonst} \end{cases}$$



Extended Color Cell Compression (XCCC)



- Erweiterung von CCC zur Verbesserung der Kompressionsrate
- Idee
 - versuche, einen großen Block mit CCC zu kodieren
 - ist die Abweichung der tatsächlichen Farbwerte im Block von a' oder b' größer als ein vorgegebener Schwellenwert, zerlege Block in vier Teilblöcke
 - verfahren rekursiv weiter



Kodierung: (a, b, 64 bit)

Kodierung: (a, b, 16 bit)

