Informationstheorie

Zusammengefasst durch Simon Flüeli

Inhaltsverzeichnis

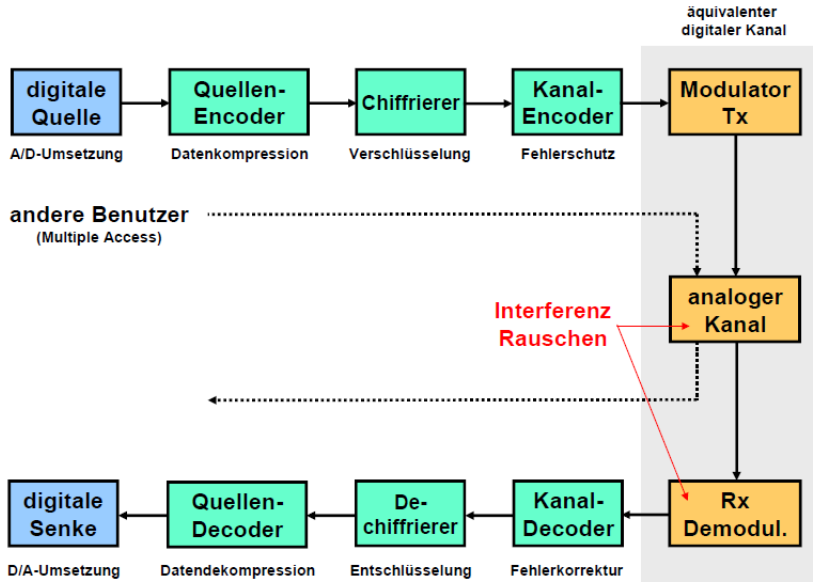
1 Kapitel 1: Einleitung

1.1 Themen der Informationstheorie

- Bei der Informationsweitergabe spricht man von Nachrichten, bei der Informationsverarbeitung von Daten
- Werden physikalisch mit Signalen dargestellt
- Seltene Ereignisse tragen viel Information
- Häufige Ereignisse tragen wenig Information
- Gründer der Informationstheorie: C. E. Shannon
  - zwei zentrale Fragen der Kommunikationstheorie
  1. Wie stark kann man Daten maximal komprimieren, ohne Information zu verlieren?
     → Eine Zufallsvariable X, z.B. ein Symbol am Ausgang einer digitalen Quelle, muss im Mittel mit **mindestens H(X) Bits** repräsentiert werden, wobei H(X) die **Entropie** oder ein Mass für die Information darstellt
     H(X) = mittlerer Informationsgehalt = E{I_x(X)} (E = Erwartungswert)
  2. Wie schnell kann man Daten "fehlerfrei" über einen verrauschten Kanal übertragen?
     Solange Datenrate R kleiner als Kanalkapazität C ist, die sich mit Hilfe der Rauschcharakteristik des Kanals bestimmen lässt, ist es theoretisch mit genügend Aufwand bei der Kanalcodierung möglich, die **Bitfehlerrate BER beliebig klein** zu machen
  3. Wie kann man Information verstecken? (nicht behandelt)

1.2 Elemente eines Übertragungssystems

- Quellencodierung, Chiffrierung, Kanalcodierung → wichtige Elemente in jedem digitalen Kommunikationssystem



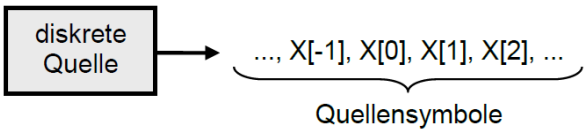
- **Hauptaufgabe der Quellencodierung:** Reduktion der Redundanz oder Irrelevanz am Ausgang der Quelle, damit weniger Daten übertragen / abgespeichert werden müssen
- **Datenkompression:** hohe wirtschaftliche Bedeutung
  - wenn weniger Daten übertragen werden müssen, kommt der Benutzer im Allgemeinen schneller in den Besitz einer Nachricht und muss weniger Übertragungszeit bzw. -volumen bezahlen
  - umgekehrt kann ein Operator mit der gleichen Infrastruktur mehr Überragungskapazität und damit mehr Einnahmen generieren

- Beispiele der Datenkompression: Kapitel 1 Seite 5, 6

2 Kapitel 2: Entropie

2.1 Diskrete Informationsquellen

2.1.1 Definition einer diskreten Nachrichtenquelle



- Erzeugt zu den diskreten Zeitpunkten t = nT die Symbole X[n].
- T = Symboldauer
- R = 1/T = Symbolrate [Symbole/s] bzw. [baud]

- Die Quellensymbole X[n] können als diskretes Zufallssignal (Zufallsprozess) oder als Folge von Zufallsvariablen aufgefasst werden
- Wir bezeichnen Zufallsvariablen mit Grossbuchstaben (z.B. X) und deren mögliche Werte oder Ereignisse (z.B. x_m) mit Kleinbuchstaben.
- Für die Wahrscheinlichkeitsverteilung von X gilt:

$$\sum_{m=1}^{M} P_X(x_m) = 1$$

- Eine Quelle, deren Symbole X[n] statistisch abhängig sind, hat ein Gedächtnis
- Gedächtnislose Quellen hingegen generieren statistisch unabhängige Symbole X[n]
- Für zwei diskrete, unabhängige Zufallsvariablen X und Y gilt: $P_{XY}(x_i, y_k) = P_X(x_i) * P_Y(y_k)$

2.1.2 Typen von gedächtnislosen Quellen

- DMS (Discrete Memoryless Source)
  Die Symbole X[n] sind unabhängig und haben identische Wahrscheinlichkeitsverteilung
- BMS (Binary Memoryless Source)
  Die unabhängigen Symbole X[n] sind 2-wertig, d.h. $P_X(x_1) = p \; und \; P_X(x_2) = 1 - p$
- BSS (Binary Symmetric Source)
  Die unabhängigen Symbole X[n] sind 2-wertig und es gilt: $P_X(x_1) = 0.5 \; und \; P_X(x_2) = 0.5$

2.2 Informationsgehalt

- Informationsgehalt eines Ereignisses $X = x_m$ ist wie folgt definiert:
$I_X(x_m) = log_2(\frac{1}{P_{XY}(x_m)})$ [bit]

- Für Ereignisse von 2 (oder mehreren) Zufallsvariablen X und Y gilt sinngemäss:
$I_{XY}(x_i, y_k) = log_2(\frac{1}{P_{XY}(x_i, y_k)})$ [bit]

- Eigenschaften:
  - **Seltene** bzw. unwahrscheinliche Ereignisse tragen **viel Information**
    z.B. Px("6er im Lotto") = (6/45)*(5/44)*(4/43)*(3/42)*(2/41)*(1/40) = 1/8145060
    Ix("6er") = 22.96 Bit wobei Ix("wieder kein 6er") = $1.77*10^{-7}$ Bit
  - Ein Symbol X trägt nur Information, wenn es mehrere Werte annehmen kann
  - Für 2 unabhängige Symbole X und Y gilt: $I_{XY}(x_i, y_k) = I_X(x_i) + I_Y(y_k)$
    Eine "6" beim Werfen des Würfels X hat den Informationsgehalt Ix("6") = log2(6) = 2.58 Bit
    Eine Doppel-"6" beim Werfen der beiden Würfel X und Y hat den Informationsgehalt
    Ixy("6", "6") = log2(1/(1/36)) = 2*log2(6) = 2*2.58 = 5.16 Bit

2.2.1 **Definition der Entropie H(X)**

- Die Entropie H(X) einer diskreten Zufallsvariablen X mit der Wahrscheinlichkeitsverteilung $P_x(x_m)$ ist wie folgt definiert:

$H(X) = \sum_{m=1}^{M} P_x(x_m) * log_2\left(\frac{1}{P_X(x_m)}\right)$ [bit] (M bei einer Münze = 2)

wobei 0*log2(0) = 0. Die Entropie H(X) ist ein Mass für die Ungewissheit der Zufallsvariablen X oder, mit anderen Worten, H(X) ist ein Mass für die Information von X.

- Die Entropie H(X) der Zufallsvariablen X ist keine Funktion von X, wie die Schreibweise nahe legt, sondern eine Funktion der Wahrscheinlichkeitsverteilung $P_x(x_m)$ von X
- Die Entropie H(X) entspricht dem mittleren Informationsgehalt $E[_{Ix}(X)]$ aller möglichen Ereignisse x_m einer Zufallsvariablen X, d.h.

$H(X) = E[I_X(X)]$

- Der Mittelwert E[X] der diskreten Zufallsvariablen X entspricht dem gewichteten Durchschnitt von X und ist gegeben durch

$$E[X] = \sum_{m=1}^{M} P_X(x_m) * f(x_m)$$

- Der Mittelwert E[f(X)] einer Funktion der diskreten Zufallsvariablen X ist gegeben durch

$$E[f(X)] = E[X] = \sum_{m=1}^{M} P_X(x_m) * f(x_m)$$

2.2.2 **Binäre Entropiefunktion**

- Beispiel Kapitel 2 Seite 5

- Die Symbole der binären, gedächtnisfreien Quelle (BMS) mit Px(0) = p und Px(1) = 1-p tragen die Information

$H(X) = h(p) = -p * log_2(p) - (1-p) * log_2(1-p)$ [bits]

- Ist **maximal**, d.h. h(p) = 1 bit, wenn die beiden Ereignisse "0" und "1" gleich wahrscheinlich sind (p=0.5)
- Ist **minimal**, d.h. h(p) = 0 wenn p =0 oder p = 1 → dann sind Quellensymbole nicht zufällig

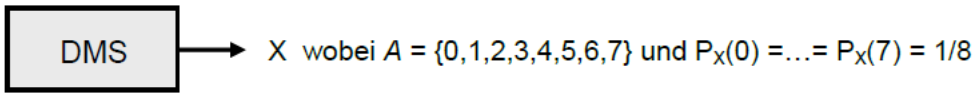
2.2.3 **Verbund-Entropie**

- Für die Verbund-Entropie H(X, Y) von 2 (oder mehreren) Zufallsvariablen X und Y gilt:

$H(X,Y) = \sum_{i,j} P_{XY}(x_i, y_j) * log_2\left(\frac{1}{P_{XY}(x_i,y_j)}\right)$ [bit]

- Die Verbund-Entropie H(X,Y) ist maximal, wen n
  - X und Y unabhängige Zufallsvariablen sind und
  - die möglichen Ereignisse von X und Y gleich wahrscheinlich sind

2.3 Minimale Anzahl binäre Fragen



- diskrete gedächtnislose Quelle (DMS), die 8-wertige Symbole X[n] im Alphabet A={0,1,2,3,4,5,6,7} generiert, wobei alle Ereignisse gleich wahrscheinlich sind ($P_X(0) = \ldots = P_X(7) = 1/8$)
- jedes Quellensymbol X trägt die Information H(X) = 3 bit (vgl. Gleichung 2.5 / 2.9 Skript 2 S. 4)

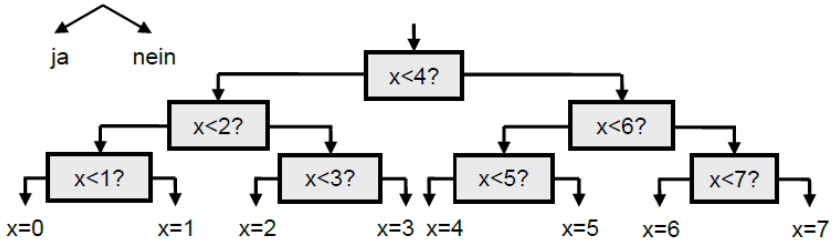
2.3.1 Strategien

2.3.1.1 Der Reihe nach durchfragen

- Bei der ersten Strategie fragt man der Reihe nach, ob es sich um die Ziffer 0, 1, 2, 3, 4, 5, 6 oder 7 handelt
  → Anzahl notwendige Fragen schwankt zwischen 1 und 7
  → Im Mittel sind 4.375 Fragen nötig (1.375 Fragen mehr als das Optimum)
    - (1+2+3+4+5+6+7)/8 = 4.375

2.3.1.2 Binärer Entscheidungsbaum

- Optimale Strategie: Man bildet den Entscheidungsbaum, indem man die Menge der möglichen Ereignisse mit einer geschickten Frage sukzessive halbiert
  → nach 3 Fragen ist man am Ziel



- gut, wenn gleichverteilt

2.4 Entropie von deutschem Text

A	B	C	D	E	F	G	H	I	J
0.04331	0.01597	0.02673	0.04385	0.14700	0.01360	0.02667	0.04355	0.06377	0.00165
K	L	M	N	O	P	Q	R	S	T
0.00956	0.02931	0.02134	0.08835	0.01772	0.00499	0.00015	0.06858	0.05388	0.04731
U	V	W	X	Y	Z	Ä	Ö	Ü	" "
0.03188	0.00735	0.01420	0.00013	0.00017	0.01423	0.00491	0.00255	0.00580	0.15149

- Die Entropie bzw. der mittlere Informationsgehalt beträgt:

$$H(X) = P(A) * log_2\left(\frac{1}{P(A)}\right) + \cdots + P(\text{" "}) * log_2\left(\frac{1}{P(\text{" "})}\right) = 4.1\frac{Bit}{Symbol}$$

- Nutzt man die häufigen Buchstabenkombinationen und Silben aus, so ergibt sich sogar nur ein mittlerer Informationsgehalt von 2.8 bit/Zeichen

- Codiert man über ganze Wörter, so trägt ein einzelnes Zeichen nur noch 2.0 bit Information
- Codiert man über ganze Sätze, so trägt ein einzelnes Zeichen nur noch 1.6 bit Information

2.5 Redundanz

- Maximale Entropie ist gegeben, wenn die einzelnen Symbole der Nachricht gleichverteilt und statistisch unabhängig sind
- Jede Abweichung der Gleichverteilung / statistischen Unabhängigkeit bewirkt eine Reduktion der Information der Nachricht
  → Redundanz R

- R einer diskreten Zufallsvariablen X mit M Werten ist die Differenz zwischen der maximalen und der tatsächlichen Entropie

$R = log_2(M) - H(X)$

- Beispiel ternäre Quelle Kapitel 1 S. 5

$R = log_2(3) - H(X) = 1.58 - 1.5 = 0.08 \frac{bit}{Symbol}$

→ wenn R > 0 kann noch komprimiert werden

3 Kapitel 3: Datenkompression

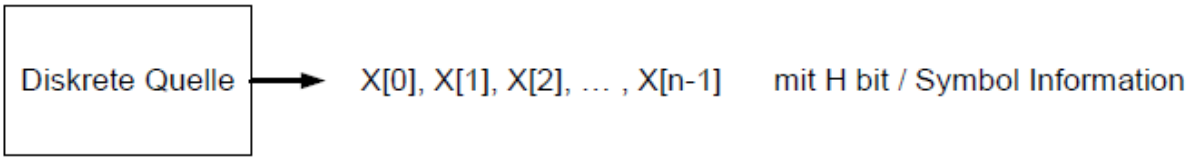
3.1 Einführung

- Datenkompression: wichtigster Teil der Quellencodierung
- Beweggründe
  - Einsparung von Speicherplatz in festen / tragbaren Geräten
  - Einsparung von Bandbreite auf Kommunikationsverbindungen
- Kompressionsverfahren
  - verlustlose Verfahren → keine Information geht verloren (nur redundante Info wird eliminiert)
    - Huffman / LZ77 / LZ78 /LZ-Welch / Lauflängencodierung (run length encoding)
    - werden angewandt, wenn nach der Dekomprimierung wieder die ursprüngliche Form ohne jegliche Abstriche vorliegen muss
  - verlustbehaftete Verfahren
    - JPEG (Joint Photographic Experts Group)
    - MPEF (Motion Pictures Expert Group)

3.2 Verlustlose Kompression

3.2.1 Das Quellencodierungstheorem

- Voraussetzung für nachfolgende Betrachtungen: diskrete gedächtnislose Quelle (DMS), die eine Folge von Symbolen als Ausgangsgrösse hat:



- Es gilt für eine Quelle ohne Gedächtnis: H = H(X)
- Für eine Quelle mit Gedächtnis ist H die Entropierate: $H = \lim_{n\to\infty} \frac{H(X[0],X[1],\ldots,H[n-1])}{n}$

3.2.1.1 Quellencodierungstheorem von Shannon

Die Quelle kann verlustlos codiert werden, solange die Coderate R grösser oder gleich der Entropierate H ist. Falls R < H beträgt, kann die Quelle auf keinen Fall verlustlos codiert werden.

- Coderate: Mittlere Anzahl Bits pro Symbol, die für die Darstellung der Symbole aufgewendet wird
- Ist Coderate = Entropierate → optimale Codierung
- Ist Coderate > Entropierate → Redundanz
- Ist Coderate < Entropierate → Quellencode erfüllt Bedingung für verlustloses Codieren nicht → Quellencode ist nicht verwendbar

3.2.2 Huffman-Codierung

- Grundidee: Häufig vorkommende Blöcke werden mit kurzen Codeworten codiert, selten vorkommende Blöcke mit längeren Codeworten
- Statische Kompressionsmethode
- Qualität hängt davon ab, wie gut das statistische Datenmodell ist

3.2.2.1 Die einzelnen Schritte

1. Ordnen der Symbole nach ihren Auftretenswahrscheinlichkeiten; sodann Symbole den Knoten des Entscheidungsbaumes zuweisen
2. Die zwei Symbole mit den kleinsten Wahrscheinlichkeiten in einem neuen Symbol zusammenfassen. Der so entstehende neue Knoten hat als Wahrscheinlichkeit die Summe der beiden (kleinsten) Wahrscheinlichkeiten
3. Falls nur noch 1 Symbole bzw. Knoten übrig bleibt → Schritt 4 durchführen, andernfalls mit Schritt 2 weiterfahren
4. Von der Wurzel aus bei jeder Verzweigung nach oben eine "0" und nach unten eine "1" eintragen

- Beispiel Kapitel 3 Seite 7

3.2.2.2 Berechnung der mittleren Codewortlänge E[L] respektive Rate R

$$R = E[L] = \sum_{m=1}^{M} P_X(x_m) * L(x_m)$$

- E[L] darf nicht kleiner als H(X) werden, da sonst Information verloren ginge
- Wenn E[L] = H(X) → Huffman-Encoder ist optimal

3.2.2.3 Blocklänge eines Huffman-Codes

- Definition: Es ist im vorliegenden Zusammenhang der "Block von Eingabesymbolen" oder Eingangszeichen: Bei zwei Eingangszeichen, die als Gesamtheit behandelt werden, ist die Blocklänge 2, usw.

- Haupteigenschaften des Huffman-Codes

Huffman-Codes sind optimal. Sie haben eine minimale mittlere Codewortlänge unter allen präfixfreien Codes

- **Nachteile** der Huffman-Codes

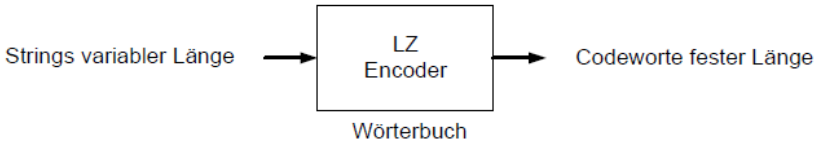
1. **Abhängigkeit von der Quellenstatistik:** Um eine Huffman-Codierung erfolgreich einzusetzen, muss die Quellenstatistik in etwa konstant bleiben. Dies ist zum Beispiel in der deutschen Sprache der Fall: In den meisten Texten wird der Buchstabe "E" mit grosser Häufigkeit (rund 14.7%) auftreten. Ergo kann diesem Umstand Rechnung getragen werden. Diese Häufigkeit kann nun für alle weiteren Buchstaben des Alphabets ermittelt werden. Wird nun aber mit dieser Statistik ein englischer Text codiert, so ist die Komprimierung weniger gut. Zuerst müsste nun eben für die englische Sprache die Quellenstatistik ermittelt werden. Unter Umständen ergeben sich unterschiede und es resultiert ein anderer Huffman-Baum.
2. **Die Quellenstatistik muss bekannt sein:** Im Fall der deutschen oder englischen Sprache (oder auch anderen Sprachen) kann die Quellenstatistik ermittelt werden. Schwieriger wird die Situation, wo dies nicht ohne weiteres gemacht werden kann. Dann müssen Annahmen getroffen werden, was wiederum erst nach mehrmaligen Korrekturen der Zuordnung von Bitfolgen zu den Symbolen zu guten Resultaten führt
3. **Komplexität:** Die Komplexität des Huffman-Baumes wächst exponentiell mit der Blocklänge n. Veranschaulichung dieses Sachverhalts: Im Beispiel mit den fünf Symbolen A, B, C, D und E (Blocklänge = 1) sollen Doppelsymbole (Blocklänge = 2) gebildet werden. Es entstehen die folgenden Doppelsymbole: AA, AB, AC, AD, AE, BA, BB, BC, … , EE, also insgesamt 25 Doppelsymbole. Der zu bildende Huffman-Baum wird aufwändig in der Verarbeitung

3.2.3 Datenkompression mit Wörterbuch nach Lempel-Ziv

- Besser als statische Methoden

3.2.3.1 Wörterbuch-basierte Kompressionsmethoden

- Symbolstrings werden mit Wörterbuch-Referenzen fester Länge codiert
- WB ist statisch oder dynamisch (bevorzugt)
- Bei Kompression sehr grosser Files → optimal bzgl. Entropie
- Bei LZ-Codierung werden Strings variabler Länge in CW fester Länge codiert
- WB wird sukzessive aufgebaut
- Kompression ist **verlustlos**, weil die ursprüngliche Zeichenkette, also die Kette der Symbole, exakt rekonstruiert werden kann



- Schritte der Grundform des LZ-Algorithmus
  1. Unterteilung der ankommenden Bitfolge in Strings variabler Länge, die sich nur in einem einzigen Bit unterscheiden
  2. Encoding eines Strings: [Position des Präfix-Strings, neues Bit]

0’ 1’ 0 0’ 0 0 1’ 1 0’ 0 0 0’ 1 0 1’ 0 0 0 0’ 0 1’ 0 1 0’ ….

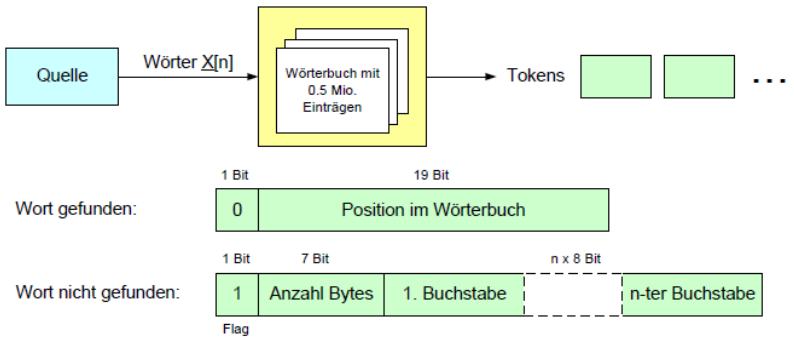
Wörterbuch-Nr:	Input	Output
1	0 → neuer String; neues Bit = 0	→ [0000 0]
2	1 → neuer String; neues Bit = 1	→ [0000 1]
3	00 → 0 gleich wie 1. String; neues Bit = 0	→ [0001 0]
4	001 → 00 gleich wie 3. String; neues Bit = 1	→ [0011 1]
5	10 → 1 gleich wie 2. String; neues Bit = 0	→ [0010 0]
6	000 → 00 gleich wie 3. String; neues Bit = 0	→ [0011 0]
7	101 → 10 gleich wie 5. String; neues Bit = 1	→ [0101 1]
8	0000 → 000 gleich wie 6. String; neues Bit = 0	→ [0110 0]
9	01 → 0 gleich wie 1. String; neues Bit =1	→ [0001 1]
10	010 → 01 gleich wie 9. String; neues Bit = 0	→ [1001 0]

- **Vorteile**
  - Codierung ist universell anwendbar. Sie ist nicht von der Quellenstatistik abhängig
  - Code ist asymptotisch optimal. Bei langen Eingangsbitfolgen kann erwartet werden, dass sich die mittlere Codewortlänge R = E[L] immer mehr (von grösseren Werten als H(X) selbst kommend) dem Wert der Entropie H(X) nähert. R bleibt im Normalfall grösser als H(X), im optimalen Fall wird R = H(X)

- **Nachteile**
  - Die Anzahl der Strings, also die Grösse des Wörterbuchs, ist beschränkt, im Beispiel auf die Zahl 15 (binär 1111)
  - Kleine Eingangsbitfolgen können kaum komprimiert werden, da am Anfang zum Aufbau des Wörterbuchs gerade das Gegenteil einer Kompression stattfindet, zum Beispiel: Für die Darstellung des ersten Eingangsbits werden fünf Ausgangsbits erzeugt

3.2.4 Datenkompression mit statischem Wörterbuch

- Wörterbuch von Anfang an statisch definiert. Neue WB-Einträge sind nicht erlaubt

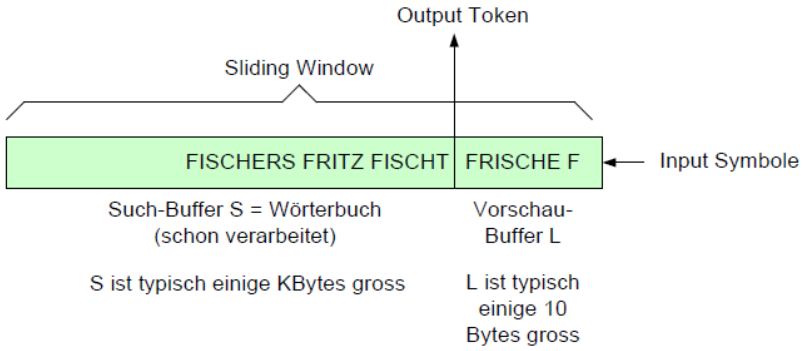


- P = Wahrscheinlichkeit, dass ein Wort im Wörterbuch gefunden wird

- Die Position im Wörterbuch wird mit 19 Bits codiert
- Wird ein Wort im WB gefunden, ist das erste Bit eine "0", dann folgt die Position im WB
- Wird ein Wort nicht im WB gefunden, so ist das erste Bit eine "1", danach folgt eine Angabe über die Anzahl Zeichen (Bytes), dann folgt Zeichen um Zeichen. Das längste mögliche Wort könnte theoretisch 127 Bytes umfassen (2^7-1)
- Die 19 Bits für die Angabe der Position im WB erlauben (2^{19}-1) = 524'287 Wörter. Das ist ein sehr umfangreiches WB

3.2.5 Datenkompression mit Wörterbuch nach LZ77

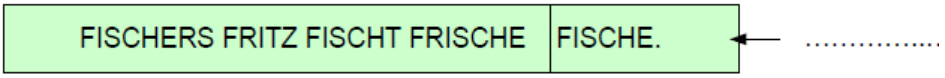
- Schiebefenster



- **Vorgehen**

1. Erstes Symbol des Vorschau-Buffers im Such-Buffer suchen, und zwar von rechts nach links. Im Beispiel ist es bei der gezeichneten Position der Leerschlag (space) vor dem Wort "FRISCHE". Nach sieben Positionen (offset = 7) befindet sich zwar ein Leerschlag, nämlich zwischen den Wörtern "FRITZ" und "FISCHT". Aber die Übereinstimmung ist länger beim Leerschlag mit offset = 13; danach kommt (nun von links nach rechts gesehen) _FRI, was bedeutet: Vier Zeichen in Übereinstimmung. (Bei offset = 7 wären es nur deren zwei, nämlich _F gewesen)
2. Das Token der längsten letzten Übereinstimmung ausgeben.
   Token = (Offset, Länge, nächstes Symbol). In unserem Fall also: Token = (13, 4, "S"); es ist das "S" im Wort "FRISCHE", denn es folgt nach den vier Zeichen _FRI. Falls es bei der Suche über die vereinbarte Suchbufferlänge keine Übereinstimmung gefunden wird, lautet jeweils der auszugebende Token = (0, 0, "nächstes Symbol")
   Das Schiebefenster um Länge+1 nach rechts verschieben, die Länge war vier, also wird das Fenster um fünf Positionen nach rechts verschoben

   Nun wird wiederum gleich verfahren: Die Suche nach "CHE_..." ergibt: Token = (23, 3, "_"): Es muss 23 Zeichen nach links gefahren werden, dort steht das "CHE" im Wort FISCHERS. Die Übereinstimmung ist 3 Zeichen, das nächste Zeichen ist ein Leerschlag

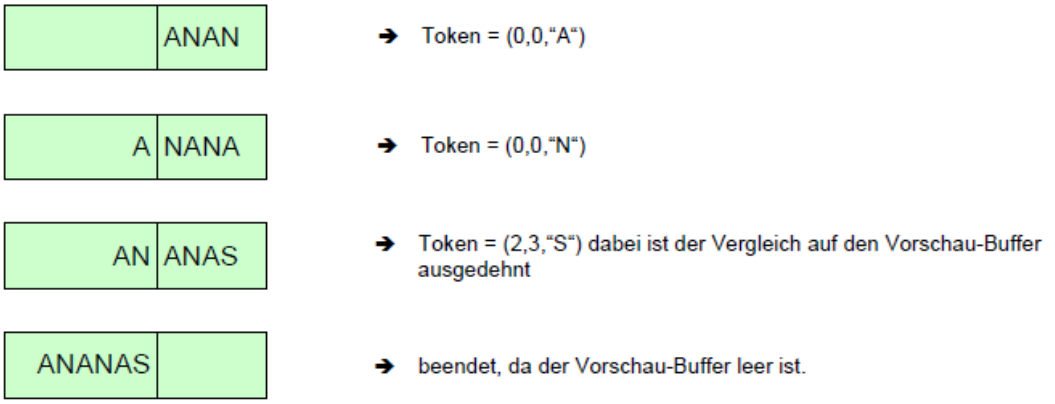


   Die Suche nach "FISCHE" ergibt: Token = (30, 6, "."). Damit ist das Ende der Zeichenkette erreicht

- **Offset S:** Anzahl möglicher Zeichen, die im Such-Buffer zurückgefahren werden darf
-**Länge L:** Länge des Vorschaubuffers

➔$[Token - Länge] = log_2(S+1) + log_2(L) + 8$

- Beispiel mit Vorschaubuffer



3.2.5.1 Anwendungen des LZ77 Verfahrens

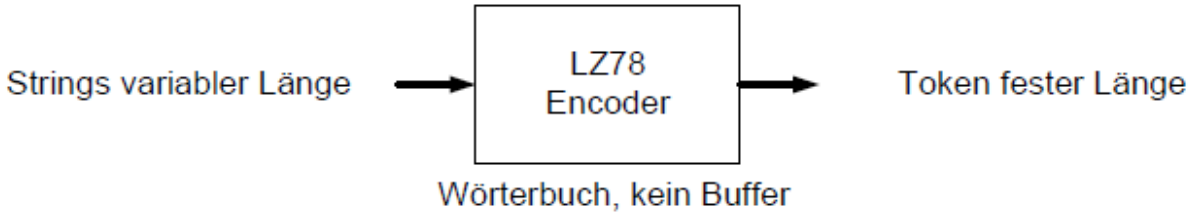
- Zip, GZip, PPP, PNG, MNG, PDF

3.2.5.2 Zusammenfassende Bemerkungen zum LZ77 Verfahren

- Der Decoder ist viel einfacher als der Encoder. Fürs Decodieren liegen alle Token vor. Durch die Angabe des Offset und dem jeweils nächsten Symbol ist kein eigentlicher Suchprozess nötig
- Daten mit nahe beieinander liegenden "Mustern" komprimieren gut, Daten mit weit auseinander liegenden "Mustern" komprimieren schlecht. Liegen die Muster zu weit auseinander, so ist der max. mögliche Offset unter Umständen zu klein → kein Bezug zu vorangehenden gleichen Muster möglich → Kompression wird verschlechtert
- Verschiedene Grössen bewirken unterschiedliches Verhalten: Grundsätzlich ermöglichen grosse Buffer eine bessere Kompression: Tokens werden dann sehr effektiv, falls die Übereinstimmung in einer grossen Anzahl Symbole stattfindet. Zum Beispiel ist ein Token1 = (17,10,"P") viel wirksamer als ein Token2 = (17,2,"S"); beim Token2 ist die Übereinstimmung lediglich 2 Zeichen, beim Token1 aber derer 10. Dem grundsätzlichen Vorteil grosser Buffer wirkt entgegen, dass der Vorschau-Buffer klein gehalten werden muss, soll die Anzahl der nötigen Vergleiche nicht zu gross werden. Denn alle möglichen Strings im Vorschaubuffer, also zum Beispiel A, AN, ANA, ANAN, müssen im Suchbuffer verglichen werden
- Die Methode kann mit statistischen Verfahren wie z.B. der Huffman-Codierung kombiniert werden. Die Token selbst werden dabei Huffman-codiert. Damit wird eine bessere Kompression erreicht

3.2.6 Datenkompression mit Wörterbuch nach LZ78

- Unterschied: ganze Buchstaben (also Mengen von 7 oder 8 Bits) oder Buchstabenkombinationen werden ins Wörterbuch aufgenommen
- Keine Buffer
- Basis ist ein WB, welches zu Beginn leer ist und dann während des Bearbeitens des zu komprimierenden Textes aufgebaut und laufend angereichert wird. Die Grösse des WB ist durch das Memory beschränkt. Ein Parser unterteilt die Symbolfolge des Textes in unterschiedliche Strings variabler Länge, die sich an den Vorgängern in lediglich **einem** Symbol unterscheiden



Der Output besteht aus einem 2-Feld-Token:

LZ78-Token = (Pointer auf einen String im WB, „neues“ Symbol)

3.2.7 Datenkompression mit Wörterbuch nach LZ-Welch

- das Symbol-Feld im Token wird eliminiert
- WB wird am Anfang mit einem definierten Alphabet initialisiert
- 8. Bit → Paritätsbit
- Der neue 1. WB Eintrag erhält die dezimale Nummer 256 (bei 8 Bit)

ASCII Zeichensatz

Dez.	Zei-chen	Dez.	Zei-chen	Dez.	Zei-Chen	Dez.	Zei-Chen	Dez.	Zei-Chen	Dez.	Zei-Chen	Dez.	Zei-chen	Dez.	Zei-chen
0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	‘	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	”	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	’	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

3.2.7.1 Der Algorithmus für die LZW-Codierung (Encoder)

0. Initialisierung: I = []
1. Neues Symbol x zu String I hinzufügen → I = Ix setzen
2. Ix im Wörterbuch verzeichnet? Falls ja, dann zu 1. , sonst zu 3.
3. a) Output = Wörterbuch-Pointer von I
   b) Neuer WB-Eintrag mit Phrase Ix
   c) I = "x" setzen

3.2.7.2 LZW-Decoder

0. WB initialisieren, normalerweise mit 256 Symbolen
1. Pointer lesen und String I ausgeben
2. Pointer lesen, String J ausgeben und erstes Symbol x von J isolieren
3. Ix im WB eintragen, Eintrag von I in "Kolonne I"; dann I = J setzen
4. Falls es noch Pointers am Eingang gibt, dann Schritt 2; sonst: Ende

3.2.7.3 Anwendungen des LZW Verfahrens

- Die "Compress" Utility der Unix-Betriebssysteme benutzt das LZW Verfahren. Dabei wird eine abgeänderte Variante des LZW Algorithmus benutzt, bei der das Wörterbuch nach Bedarf wachsen kann. (Utilities sind Werkzeuge, die den Umgang mit dem Rechner erleichtern. Beispiel: Der Benutzer wird bei grundlegenden Aufgaben unterstützt, z.B. beim Kopieren von Daten).
- Bei Bildern im GIF Format
- Ausserdem wird das LZW Verfahren bei Modems nach V.42 angewandt. Es existieren zwei Modi: Der "transparent mode", der ohne Kompression arbeitet und der "compressed mode", der mit einer Variante des LZW Verfahrens arbeitet

3.2.8 Run Length Encoding (Lauflängencodierung) für Texte

- "run" = Anzahl identischer Zeichen, die aufeinander folgend auftreten. Es ist der Repetitionscounter

3.2.8.1 Grundidee

- Zeichenketten mit Folgen gleicher Zeichen wie zum Beispiel d … d sind zu ersetzen durch (n,d). Dabei ist n der Zähler, der die Anzahl angibt, d ist das Datensymbol. In diesem Sinn wird also die Symbolfolge ddddddddd zu (9,d); der ‚run" beträgt neun. Eine andere Symbolfolge SSSSSSSSSSS mit ‚run" zwölf wird zu (12,S).
Wie können nun auch Zahlen dieser Prozedur unterzogen werden, ohne dass die Zahl selbst mit dem Repetitionscounter verwechselt wird? Eine Variante ist, dem Repetitionscounter einen ‚escape character" voranzustellen. So entstehen dann für die obigen Beispiele die folgenden Ausdrücke:
ddddddddd wird zu @9d
SSSSSSSSSSS wird zu @12S
- Mit der **Notation (escape character, counter, data),** die drei Bytes vorsieht, könnten auf diese Weise Folgen mit bis zu 255 gleichen Zeichen erfasst werden. Dabei sollten nur **runs >3** durch die Notation (escape character, counter, data) ersetzt werden, da erst dann eine Kompression resultiert.
Wie kann der ‚escape character" selbst geschrieben werden, ohne dass er als Ankündigung eines ‚runs" dasteht? Eine Möglichkeit besteht darin, das @-Zeichen durch (@,1,@) zu ersetzen. In diesem Fall kann für die Darstellung des „@" natürlich von Kompression keine Rede mehr sein.
Gemäss den obigen Ausführungen wird aus der Zeichenfolge
AAAAAABBBAAAACAABBBBBBBBBCCCCC…
nun also
@6ABBB@4ACAA@8B@5C …

3.2.9 Run Length Encoding (RLE) für Bilder

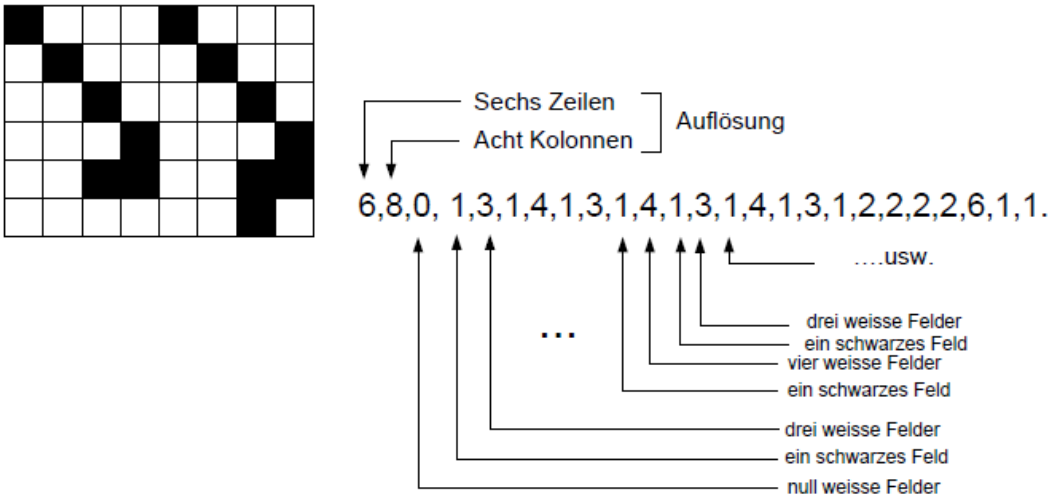
Falls benachbarte Pixel die gleiche Farbe bzw. Intensität aufweisen, kann dieser Umstand genutzt werden; es wird eine mehr oder weniger grosse Kompression erreicht. Weil bei Bildern oftmals grössere Bildausschnitte mit gleichem Farbton oder gleicher Graustufe auftreten, können unter Umständen grosse ‚runs" ausgenutzt werden.
Die Pixelgrösse weist folgende Anzahl Bits auf:

- Schwarz-weiss-Bilder: 1 Bit pro Pixel. Damit kann die Unterscheidung schwarz / weiss getroffen werden
- Graustufenbild mit 256 Stufen: 8 Bit pro Pixel
- RGB Bild mit drei Farben Rot, Grün, Blau: 1 Byte pro Farbe, also 24 Bit pro Pixel

3.2.9.1 RLE-Kompression am Beispiel eines Schwarz-weiss-Bildes

Am Anfang wird die Bildgrösse spezifiziert, also 6 Zeilen, 8 Kolonnen.

Annahme:
Es liegt die Vereinbarung vor, dass bei der Aufzählung mit den weissen Feldern gestartet wird.



Es ist auch eine zeilenweise Codierung möglich, zum Beispiel wie folgt:

Zeile 1: 0,1,3,1,3,eol
Zeile 2: 1,1,3,1,2,eol
Zeile 3: 2,1,3,1,1,eol
Zeile 4: 3,1,3,1,eol
Zeile 5: 2,2,2,2,eol
Zeile 6: 6,1,1,eol

3.2.9.2 Kompression von Graustufen-Bildzellen

Beispiel:

Gegeben seien die folgenden Graustufenwerte:

12,12,12,12,12,12,12,12,12,35,76,112,67,87,87,87,87,5,5,5,5,5,5,1, …

Falls die Zahl 255 als Escape-Character definiert ist, kann damit die folgende Konvention getroffen werden:

(escape character, counter, data) = (255, counter, data)

Damit gilt für die Graustufenwerte im Beispiel die folgende Notation:

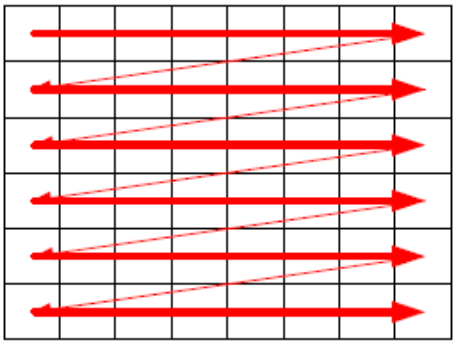
255,**9**,12,35,76,112,67,255,**4**,87,255,**6**,5,1, …

Hierin stehen die fett gedruckten Werte 9, 4 und 6 für Counters. Durch die Verwendung des Werts 255 als Escape-Character kann eine Graustufe weniger dargestellt werden.

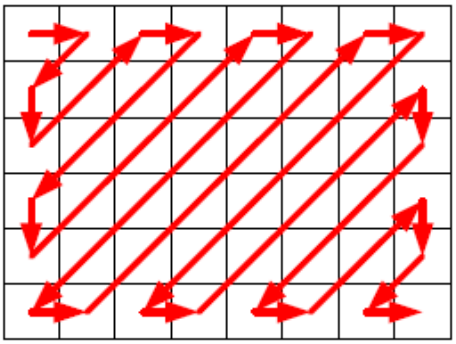
Abtastung der Werte (RLE-Scanning)

Es gibt verschiedene Möglichkeiten für das Scanning:

Für Telefaxe:



Für JPEG:



3.3 JPEG als verlustbehaftetes Verfahren für die Bildkompression

3.3.1 Einleitung

- Der Verlust von Bildauflösung wird bis zu einem gewissen Grad akzeptiert

3.3.2 Die wichtigsten 7 Schritte des JPEG-Verfahrens

- **Schritt 1: Transformation der Farbbilder in eine Darstellung mit Luminanz und Chrominanz**
  Menschliches Auge reagiert empfindlicher auf Helligkeitsunterschiede, als auf farbliche Differenzen → Farbinformation höher komprimieren als Helligkeitsinformation
  ➔ Dieser Schritt, also die Aufteilung eines Farbbildes in einen Teil Luminanz und einen Teil Chrominanz, ist eine Vorbereitung für die eigentliche Kompression

- **Schritt 2: Downsampling der beiden Chrominanz-Komponenten**
  Farbinformationen werden komprimiert. Üblich sind Downsampling-Raten von 2:1 horizontal und vertikal (2h2v oder 4:1:1). Die Bildgrösse wird damit zu: $\frac{1}{3} + \left(\frac{2}{3}\right) * \left(\frac{1}{4}\right) = \frac{1}{2}$ (Siehe Seite 26)
- **Schritt 3: Pixel-Gruppierung der Farbkomponenten in 8x8 Blöcke**
  Hierbei wird die horizontale und vertikale Korrelation ausgenützt. Die Blöcke werden separat

komprimiert. Dies stellt allerdings eine Schwachstelle des Verfahrens dar, denn das Optimum liegt nicht zwingend bei einer Blockgrösse von 8x8 Pixeln. Diese Blockgrösse hat eher historischen Hintergrund und hat mit den Wortbreiten der Prozessoren zu tun, die zur Zeit der Entwicklung von JPEG aktuell waren

- **Schritt 4: Diskrete Cosinustransformation (8x8 DCT)**
  Die Werte werden nun in den Frequenzbereich transformiert. Hier geschieht die eigentliche Vorbereitung für die Datenkompression. Es stellt sich nämlich heraus, dass einzig der DC-Wert und einige wenige (drei bis vier) tieffrequente AC-Werte bereits genügend Bildinformation enthalten, um den ganzen 8x8-Pixelblock darzustellen. Der Begriff "tieffrequent" bezieht sich dabei auf Ortsfrequenzen

- **Schritt 5: Individuelle Quantisierung einzelner Frequenzkomponenten**
  Das Prinzip ist hier das folgende: Die Frequenzkomponenten mit viel Bildinformation werden fein quantisiert. Die Komponenten mit wenig Bildinformation werden grob quantisiert. Dadurch geschieht in der praktischen Umsetzung folgendes: die meisten grob quantisierten Komponenten erhalten den Wert Null

- **Schritt 6: Entropiecodierung der quantisierten Frequenzkomponenten**
  Hier erfolgt eine Entropiecodierung der Werte, die erhalten bleiben. Dazu ist zu bemerken: Eine Entropiecodierung kann immer versucht werden. Sie ist dann aussichtsreich, falls die Werte nicht die gleiche Häufigkeit haben. Die Entropiecodierung ist immer verlustlos; dieser Teil in der ganzen Kette der Aktivitäten bewirkt also keinerlei Verlust von Information. Als sinnvolle Verfahren der Entropiecodierung innerhalb JPEG haben sich die Lauflängencodierung und die Huffmancodierung bewährt. Es wird eine Kombination beider Verfahren angewandt

- **Schritt 7: Hinzufügen von Header und JPEG Parameter**
  Es werden die folgenden Angaben mitgeliefert

Offset	Grösse in Bytes	Beschreibung
0	2	JPEG Start of Image (SOI) Marker ($FFD8_H$)
2	2	Bildbreite in Pixel
4	2	Bildhöhe in Pixel
6	1	Anzahl Komponenten (1: Graustufenbild; 3: RGB-Bild)
7	1	Horizontale/vertikale Downsampling-Rate für Komponente 1
8	1	Downsampling-Rate für Komponente 2 (bei RGB-Bildern)
9	1	Downsampling-Rate für Komponente 3 (bei RGB-Bildern)

Die Komponente 1 entspricht der Luminanz, die Komponenten 2 und der Chrominanz Blau C_B und Chrominanz Rot C_R.

3.3.3 Das Luminanz / Chrominanz Farbmodell

- Die drei Farbanteile Rot, Grün, Blau werden in die folgenden Anteile umgerechnet
Y: Luminanz, das heisst die Graustufenintensität
C_B: Blauanteil der Chrominanz
C_R: Rotanteil der Chrominanz

Es gelten die folgenden Beziehungen (3.4):

Luminanz / Chrominanz Farbmodell:

$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

Die Farbkomponenten Rot, Grün und Blau können wie folgt notiert werden; Gleichungssystem (3.5):

Berechnung der R-, G- und B-Komponenten

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ C_B - 128 \\ C_R - 128 \end{bmatrix}$$

→ Um die Helligkeit (Luminanz Y) zu beschreiben, sind drei Werte notwendig: Ein Anteil rot, ein Anteil grün und ein Anteil blau. Wie Werte 0.299, 0.587 und 0.114 stellen die Intensitätswerte der drei Farbanteile dar, die für die Berechnung der Luminanz eingehen

- Um den Farbanteil (Chrominanz B) zu beschreiben, sind ebenfalls drei Werte notwendig: Ein Anteil Rot, ein Anteil Grün und ein Anteil Blau
- Um den Farbanteil (Chrominanz R) zu beschreiben, sind ebenfalls drei Werte notwendig: Ein Anteil Rot, ein Anteil Grün und ein Anteil Blau

Setzt man gemäss (3.5) für $R = Y + 1.402 \cdot (C_r - 128)$ erhält man:

$$R = 0.299R + 0.587G + 0.114B + 1.402 \cdot (0.5R - 0.4187G - 0.0813B + 128 - 128) = R$$

was zu erwarten war. Die dazu analoge Kontrollrechnung lässt sich auch für G (grün) und B (blau) durchführen. Damit wird nichts anderes als die Übereinstimmung der beiden Gleichungssysteme (3.4) und (3.5) geprüft.

3.3.4 Downsampling der Chrominanz-Komponenten

Ein 4:2:2 Downsampling der Chrominanz-Komponenten in einem 16x16 Bild

Gegeben ist ein Bild mit 16 x 16 = 256 Pixel. Daraus ergeben sich 256 Werte für die Luminanz.

Für die Chrominanzen C_B und C_R wird nun jeweils der lineare Mittelwert zweier benachbarter Pixel berechnet. Es handelt sich um folgende Art des Downsamplings:

2:1 horizontal und 1:1 vertikal (2h1v oder 4:2:2)

Die Bildgrösse wird damit zu: $1/3 + (2/3) \cdot (1/2) = 2/3$

Downsampling 16 x 16 Bild:

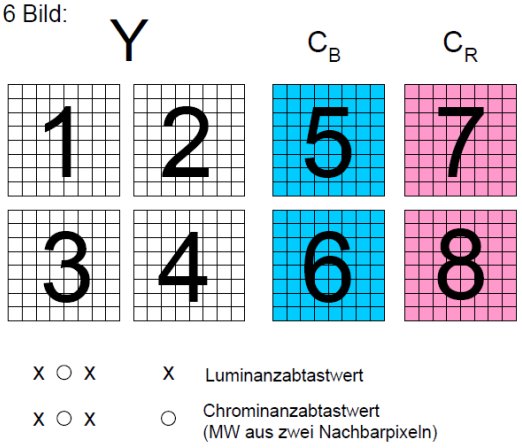


Abb. 3.17: Luminanz- und Chrominanzabtastwerte (4:2:2)

Die Luminanzabtastwerte werden also unverändert übernommen (keine Kompression).

Ein 4:1:1 Downsampling der Chrominanzkomponenten in einem 16x16 Bild

Wiederum kommt für die Chrominanzkomponenten eine Mittelwertbildung ins Spiel.

Gegeben ist wieder das Bild mit 16 x 16 = 256 Pixel. Daraus ergeben sich 256 Werte für die Luminanz.

Für die Chrominanzen C_B und C_R wird nun jeweils der lineare Mittelwert von *vier* benachbarten Pixeln berechnet. Es handelt sich um folgende Art des Downsamplings:

2:1 horizontal und vertikal (2h2v oder 4:1:1)

Die Bildgrösse wird damit zu: $1/3 + (2/3)\cdot(1/4) = 1/2$

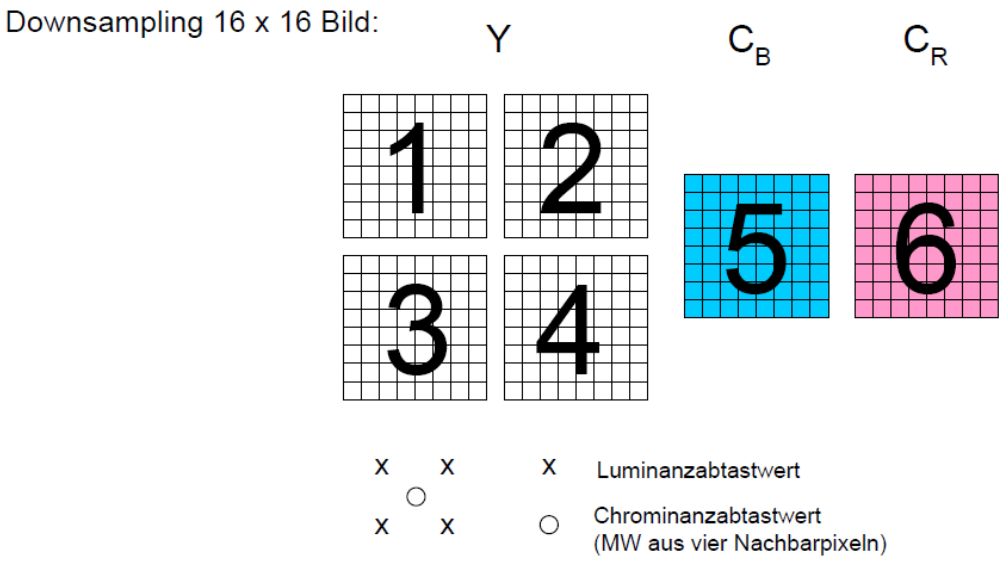


Abb. 3.18: Luminanz- und Chrominanzabtastwerte (4:1:1)

Die Luminanzabtastwerte werden wiederum unverändert übernommen (keine Kompression).

3.3.5 Die JPEG Blockverarbeitung

- Alle Blöcke zu 8x8 Pixel werden nun wie folgt weiter verarbeitet:
- **Zweidimensionale diskrete Cosinustransformation:** Hier werden die Informationen, die als Funktion des Orts vorliegen, in Informationen in Form von Amplituden zu Ortsfrequenzen gewandelt. Wozu soll dies gut sein? Es stellt sich heraus, dass dies eine sinnvolle Massnahme ist. Würden die Werte im Ortsbereich quantisiert werden, so erhielte man sehr viel mehr Werte mit signifikanten Grössen, als wenn dies im Ortsfrequenzbereich vorgenommen wird
- Die zweidimensionale Cosinustransformation entspricht einer Transformation der Achsen des Bezugssystems, also des Koordinatensystems, in dem die entsprechenden Werte aufzutragen sind. *Vor* der Transformation liegen viele Punkte an Orten, wo sich grosse Werte von m und n ergeben; **m und n sind Ortskoordinaten**. *Nach* der Transformation in den Ortsfrequenzbereich, also ins neue Achsensystem mit den Ortsfrequenzachsen u und v ist zu erkennen, dass die Punkte in ein Gebiet nahe bei der einen Achse zu liegen kommen. Die Werte **u und v sind Ortsfrequenzen**. Der Umstand, dass die Werte nahe bei der einen Achse zu liegen kommen, bedeutet, dass der eine Wert des Wertepaars (u, v) eines Punktes jeweils sehr klein ist und unter Umständen vernachlässigt werden kann. Somit ergeben sich nach der Quantisierung viel weniger Werte, die berücksichtigt werden müssen.

- **Quantisierer:** Hier wird eine Quantisierung vorgenommen. Diese trägt, zusammen mit der Cosinustransformation, wesentlich zur datenkompression bei, weil bei sehr vielen der insgesamt 64 Werten der Ortsfrequenzen nach der Quantisierung der Wert Null übrig bleibt

- **Entropy Encoder (Huffman):** Die übrig bleibenden relevanten Werte werden nun noch mit einem Entropieverfahren codiert. Dies steuert nochmals einen kleinen Beitrag zur gesamten Datenkompression bei. Dieser Teil der Kompression ist gänzlich verlustlos, weil Entropieverfahren verlustlos sind

- **Compressed Data:** Die im JPEG-Format komprimierten Daten

Blockdiagramm der JPEG Verarbeitung:

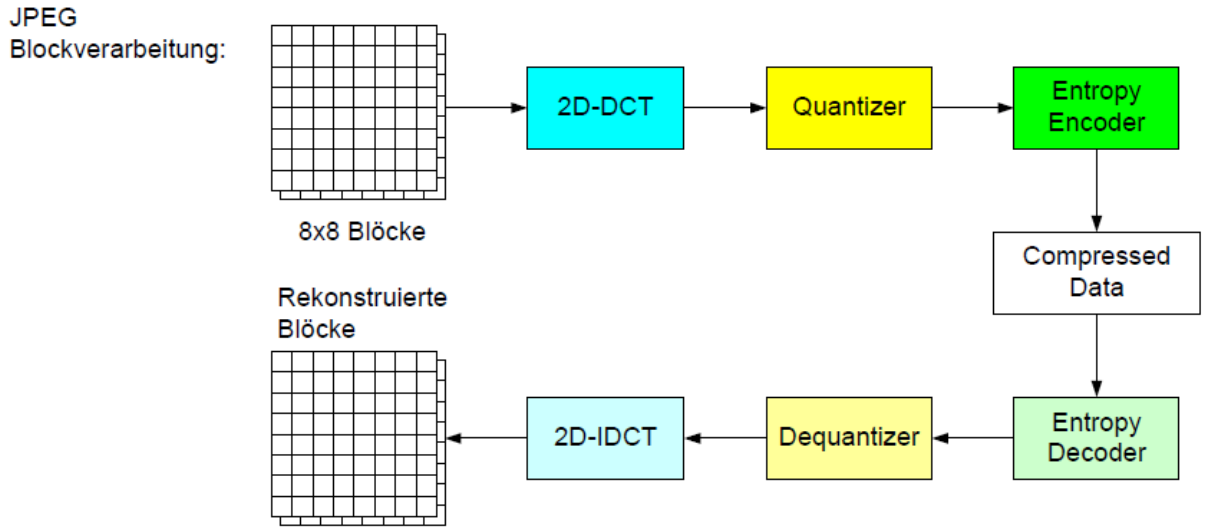


Abb. 3.19: JPEG Blockverarbeitung

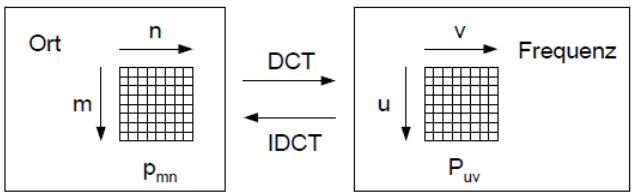
3.3.6 Die zweidimensionale diskrete Cosinustransformation

- Formeln mit den Werten für M=N=8 und nachfolgend Beispiel anhand einer 2x1 DCT

MxN DCT (JPEG: M=N=8)

$$P_{u,v} = \frac{2}{\sqrt{M*N}} * C(u) * C(v) * \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} p_{mn} * \cos\left(\frac{(2m+1)*u*\pi}{2M}\right) * \cos\left(\frac{(2n+1)*v*\pi}{2N}\right)$$

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & u, v = 0 \\ 1 & sonst \end{cases}$$



MxN inverse DCT (JPEG: M=N=8)

$$p_{mn} = \frac{2}{\sqrt{M*N}} * \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} C(u) * C(v) * P_{uv} * \cos\left(\frac{(2m+1)*u*\pi}{2M}\right) * \cos\left(\frac{(2n+1)*v*\pi}{2N}\right)$$

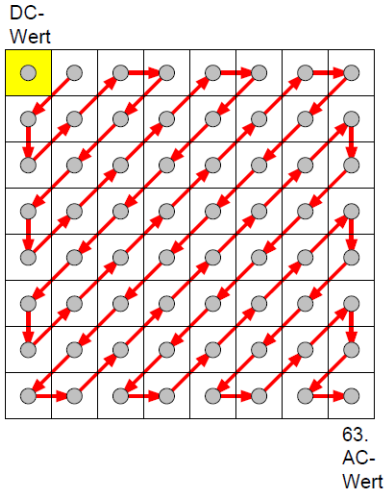
3.3.7 Bildpunkte, DCT-Koeffizienten, Quantisierung, Rekonstruktion, Zick-Zack-Scanning und Entropiecodierung für einen 8x8-Pixelblock

TODO
TODO
TODO
TODO
TODO
TODO
TODO
TODO
TODO
TODO
TODO

3.3.7.1 Das Zick-Zack-Scanning der DCT-Koeffizienten

- Weil die signifikanten Werte nahe bei der linken oberen Ecke des Array liegen, ist es sinnvoll, die Abtastung entlang dieser Werte durchzuführen

DCT-Koeffizienten: Zick-Zack-Scanning



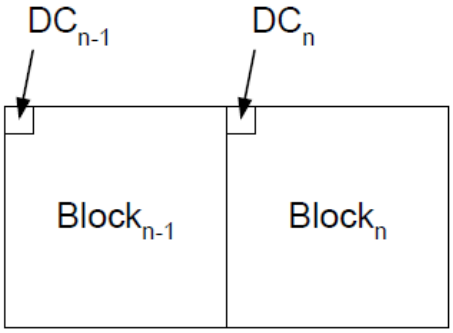
- Dabei wird eine Lauflängencodierung der aufeinander folgender Nullen vorgenommen. Das "End of Block" (EOB) Symbol steht für "alles Nullen bis zum Blockende"

3.3.7.2 Entropiecodierung des DC-Koeffizienten

- Es wird eine horizontale Prädiktion verwendet

Darstellung der DC-Koeffizienten

Horizontale Prädiktion:
$DC_n - DC_{n-1}$



- Darstellung der codierten Werte:
  - **Symbol 1:** Anzahl Bits, die für die Angabe der Amplitude verwendet werden
  - **Symbol 2:** Amplitude

3.3.7.3 Entropiecodierung der AC-Koeffizienten

- Es kommt das folgende Verfahren zum Einsatz
  - **Symbol 1:** Anzahl Nullen vor dem Koeffizienten, danach die Anzahl Bits, die für die Angabe der Amplitude verwendet werden
  - **Symbol 2:** Amplitude, angegeben mit einem Code variabler Länge

Beispiel:

quantisierte Koeffizienten

$P'_{uv} = \text{round}\,(P_{uv}/Q_{uv})$

Annahme bei diesem Beispiel: Der vorangegangene DC-Wert betrug 12. ➔ der neue Wert 15 ist also um den Wert 3 grösser

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-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

Abb. 3.29: Quantisierte Koeffizienten

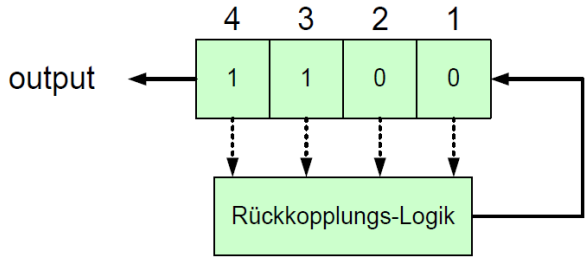
4 Kapitel 4 PN-Sequenzen

4.1 Einführung

- Pseudo-Noise Sequenzen (künstlich erzeugte Sequenzen, die die Eigenschaften von "coin-flipping"-Sequenzen fast erfüllen
- Um mit geringem Aufwand eine technisch "brauchbare" Zufallssequenz zu erzeugen, braucht man **rückgekoppelte Schieberegister (Linear Feedback Shift Register LFSR)**
  → Draus entsteht eine Pseudozufallssequenz
  → Bitfolgen sind periodisch

4.2 Linear Feedback Shift Register

- Beispiel 4-Stufiges (n=4) Schieberegister mit Rückkopplung



- Im Allgemeinen haben n-stufige LFSR-Sequenzen die Periode $P \leq 2^n - 1$

4.2.1 Anwendungen von LFSR

- Testdatenquelle für verschiedenste Zwecke
- Erzeugung von Zufälligkeit in stochastischen Modellen, Beispiel: Monte Carlo-Simulation. Die Grundlage sind Zufallsexperimente. Dabei ist es das Ziel, aufgrund der Resultate mittels der Wahrscheinlichkeitstheorie analytisch nicht lösbare Probleme numerisch zu lösen. *Wie erhält man die benötigten Zufallsexperimente?* Genau hier setzt die Anwendung von Pseudozufallssequenzen an
- Scrambling eines Bitstromes, um die Störstrahlungseigenschaften zu verringern sowie zur Gewährleistung eines gleichstromfreien Signals
- Ranging (Entfernungsmessung) beim GPS
- Bandspreizsysteme bei denen CDMA eingesetzt wird (Anwendung beim UMTS)
- Bescheidene Zufallsgeneratoren

4.3 m-Sequenzen

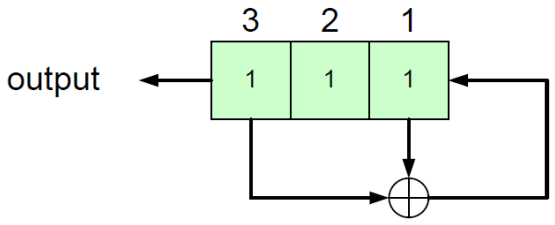
- PN Sequenzen mit **maximal möglicher Länge**
- Falls Sequenzen die maximal mögliche Länge aufweisen, wiederholen sie sich mit einer Periode von $P = 2^n - 1$ (n = Anzahl Flip-Flops)
- Damit m-Sequenz entsteht, müssen Abgriffe für die EXOR-Verknüpfung einem Feedback-Polynom entsprechen, das mathematisch "primitiv" ist
  - Dieses primitive Polynom **modulo 2 der Ordnung n** definiert eine Rekursionsbeziehung, um ein nächstfolgendes Zufallsbit zu erhalten

Beispiel mit n = 3 Stufen

- Gesucht: Rückkopplungen, um eine m-Sequenz zu erhalten
- Weil n = 3, können wir die Periode berechnen, die wir von der m-Sequenz erwarten:

$$P = 2^3 - 1 = 7$$

- Das primitive Polynom lautet (3, 1, 0) ➔ Siehe Extrablatt Seite 299
  - Das Feedback-Polynom lautet: $x^3 + x^1 + 1$



← Zugehörige Schaltung

- Bei der Schaltung wurde ein Anfangszustand (seed) von [1 1 1] eingesetzt. Es ist jedem der Anfangszustände [0 0 1], [0 1 0], [0 1 1], [1 0 0], [1 0 1], [1 1 0] auch erlaubt, als seed zu wirken
- Der seed sei also [1 1 1]

[1 1 1] führt zum Ausgang des EXOR: ➔ ***0***; dann Verschiebung um ein Bit ➔ [1 1 0]
[1 1 0] führt zum Ausgang des EXOR: ➔ 1; dann Verschiebung um ein Bit ➔ [1 0 1]
[1 0 1] führt zum Ausgang des EXOR: ➔ 0; dann Verschiebung um ein Bit ➔ [0 1 0]
[0 1 0] führt zum Ausgang des EXOR: ➔ 0; dann Verschiebung um ein Bit ➔ [1 0 0]
[1 0 0] führt zum Ausgang des EXOR: ➔ 1; dann Verschiebung um ein Bit ➔ [0 0 1]
[0 0 1] führt zum Ausgang des EXOR: ➔ 1; dann Verschiebung um ein Bit ➔ [0 1 1]
[0 1 1] führt zum Ausgang des EXOR: ➔ 1; dann Verschiebung um ein Bit ➔ [1 1 1]
und wiederum:
[1 1 1] führt zum Ausgang des EXOR: ➔ 0; dann Verschiebung um ein Bit ➔ [1 1 0]
Output: 1 1 1 ***0*** 1 0 0 1 1 1 0 ….
Die Folge wiederholt sich nach 1 1 1 0 1 0 0; also lautet die Periode: 1 1 1 0 1 0 0

X-OR Wahrheitstabelle:

A	B	Y = A ⊻ B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Die Zustandsfolge (dezimal) lautet: 4, 1, 3, 7, 6, 5, 2, dann wieder 4, ...
1 1 1 → 7
1 1 0 → 6
1 0 1 → 5
0 1 0 → 2 etc.

➔ **Resultat: Das gegebene dreistufige Schieberegister liefert eine Maximallängensequenz der Länge $2^n - 1 = 2^3 - 1 = 7$, wie zu erwarten war, aufgrund der Anwendung der Rückkopplung**

4.3.1 Zufallseigenschaften der m-Sequenzen

- Die m-Sequenzen sind PN -Sequenzen, weil sie die folgenden drei Bedingungen für PN-Sequenzen erfüllen

1 m-Sequenzen sind fast ausgeglichen in Bezug auf die Anzahl Einsen und Nullen
2 Die relative Häufigkeit von runs der Länge k beträgt $\frac{1}{2^k} f\ddot{u}r\ k \leq (n-1)\ und \frac{1}{2^{k-1}}\ f\ddot{u}r\ k = n.$ Ein "run" ist das Aufeinanderfolgen mehrerer Nullen und Einsen. So weist zum Beispiel die Bitfolge …00110101000111001101… folgende runs auf: Run 1 kommt 6x vor, run 2 kommt 4x vor und run 3 kommt 2x vor
3 Die m-Sequenz der Länge P und die zyklisch verschobene Kopie haben fast 50% übereinstimmende Bits und 50% verschiedene Bits

m-Sequenzen sind PN-Sequenzen
- Maximallängensequenzen sind PN-Sequenzen (erfüllen Eigenschaft einer pseudozufälligen Folge)
- eine PN-Sequenz ist **nicht immer** eine Maximallängensequenz

m-Sequenzen sind fast ausgeglichen in Bezug auf die Anzahl Einsen und Nullen
- Bei 5-stufigem Register: m-Sequenz von

$$1\ 1\ 1\ 1\ 1\ \boldsymbol{0}\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0 \mid\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1 \ldots$$

→ 15 Nullen und 16 Einsen
→ Anzahl "1" ist stets um 1 grösser als Anzahl "0"

- Für sehr grosse Perioden P gilt:

$$\Pr(0) = \Pr(1) \Rightarrow 0.5$$

- Berechnung der Wahrscheinlichkeitsverteilung für die "0" und "1" in Abhängigkeit der Periode P: (P = Anzahl Bits, bis Wiederholung)

$$\Pr(1) = \frac{0.5 * (P+1)}{P} = 0.5 * \left(1 + \frac{1}{P}\right)$$
$$\Pr(0) = 0.5 * \left(1 - \frac{1}{P}\right)$$

Die relative Häufigkeit von "runs" der Länge k beträgt $\frac{1}{2^k} f\ddot{u}r\ k \leq (n-1)\ und \frac{1}{2^{k-1}}\ f\ddot{u}r\ k = n$
- Siehe Skript Seite 8

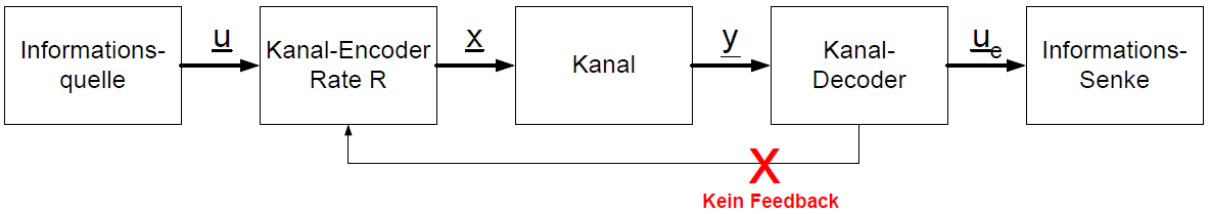
Die m-Sequenz der Länge P und die zyklisch verschobene Kopie haben fast 50% übereinstimmende Bits und 50% verschiedene Bits
- Folgende Haupteigenschaften
  - Die Hammingdistanz zwischen der m-Sequenz s und der zyklisch um einen Rechts-Shift verschobenen Sequenz beträgt 2^{n-1}
  - Es gilt eine "Shift-and-Add-Eigenschaft", die folgendes besagt: Die Modulo 2 Addition der Sequenz s mit der verschobenen Sequenz $T^u\{s\}$ führt zu einer (ebenfalls verschobenen) Sequenz $T^v\{s\}$. Aber es ist wiederum die Sequenz s, eben nur verschoben

- Siehe Skript Seite 9

5 Kapitel 5: Kanalcodierung (Teil 1)

5.1 Einführung

- Hauptziel: Daten zuverlässig von der Quelle bis zur Senke zu übertragen
- oftmals nur durch zusätzliche Fehlerschutzbits erreichbar



- Decoder muss selbstständig entscheiden können, wie die richtige (ursprünglich) gesendete Bitfolge lautet (darum kein Feedback)

- Endziel: Daten erfolgreich und fehlerfrei von der Quelle zur Senke transportieren
→ zwei Vorgehensweisen
- Erste Variante: Es wird über einen bestimmten Datensatz, zum Beispiel ein Datenrahmen, eine Prüfsumme gebildet. Stimmen die Prüfsummen beim Sender und Empfänger überein, kann von einer fast 100%-ig fehlerlosen Übertragung ausgegangen werden. Der Sender erhält ein "Acknowledge" (ACK). Stimmen die Prüfsummen nicht überein, so bleibt der ACK aus. Nach einer bestimmten Zeit des Ausbleibens des ACK, wird der Datensatz nochmals gesendet (Retransmission), bis die Prüfsummen schliesslich übereinstimmen (Beispiel TCP, Transmission Control Protocol)
- Zweite Variante: Es ist eine "Forward Error Correction" angewandt. In den Sendebitstrom werden zusätzliche Bits eingeschleust, um damit beim Empfänger einfache oder mehrfache Bitfehler korrigieren zu können. Ein Wiedersenden (Retransmission) ist nicht vorgesehen. Der Sendebitstrom mit den eingeschleusten zusätzlichen Bits bietet die Grundlage, damit der Empfänger bis zu einem gewissen Grad Fehler selbst zu korrigieren vermag

- Bei der Forward Error Correction geht es darum, die zu sendenden Daten so aufzubereiten, dass eine bestimmte Anzahl von Bitfehlern vom Empfänger selbst korrigiert werden können

5.2 Der Kommunikationskanal

- Definition: Diskreter Kanal
- Wenn in einem zeitdiskreten Kanal die Werte, welche Eingangs- und Ausgangsvariablen einnehmen können, endlich oder zählbar unendlich sind, bezeichnet man den Kanal als diskreten Kanal

5.2.1 Das Kanalcodierungstheorem nach Shannon

- Kanalkapazität eines diskreten gedächtnisfreien Kanals

$$C = max_{P(x)}\,[H(Y) - H(Y|X)]$$

H(Y) = Erwartungs-/Mittelwerd von $I_y(Y)$
H(Y|X) = Fehlerfall
- Einheit von C = [C] = Bit/Kanalbenützung → Kanalkapazität

$$Coderate\ R = \frac{Anzahl\ Infobits}{Anzahl\ Codebits} = \frac{K}{N}$$

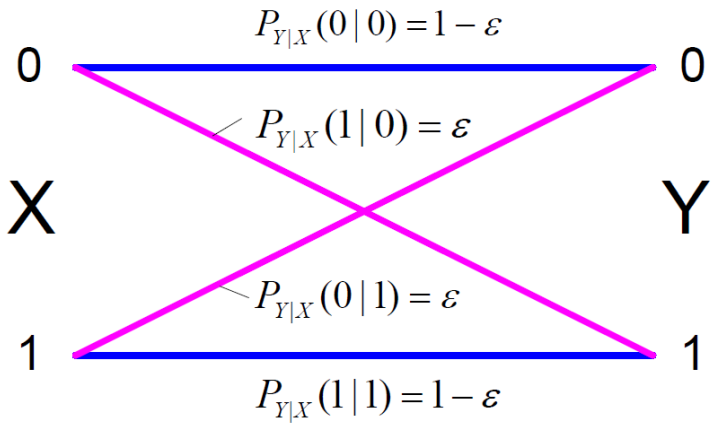
- Falls die Übertragungsrate **R < C** beträgt, existiert ein Code mit ausreichend grosser **Blocklänge N**, mit dem die Fehlerwahrscheinlichkeit beliebig klein gemacht werden kann
- Falls **R > C** beträgt, so ist die Fehlerwahrscheinlichkeit von jedem Code immer > 0, sei N noch so lang

- Eigenschaften

- Rauschen im Kanal beschränkt nicht die Zuverlässigkeit der Übertragung, sondern hauptsächlich die Übertragungsrate
- Verschiedene Kanäle lassen sich auf diese Weise mit je einer Zahl vergleichen
- Je grösser die Blocklänge N des verwendeten Block-Codes ist, desto komplexer wird der Decoder
- Das Kanalcodierungstheorem gibt Bedingungen für die Existenz von Kanalcodes an. Das Theorem liefert jedoch keine Algorithmen

5.2.2 Der Binary Symmetric Channel (BSC)

- Oft für die Modellierung digitaler Nachrichtensysteme benutzt
- Gedächtnisfrei
- ε → Übergangswahrscheinlichkeit (Fehlerwahrscheinlichkeit)



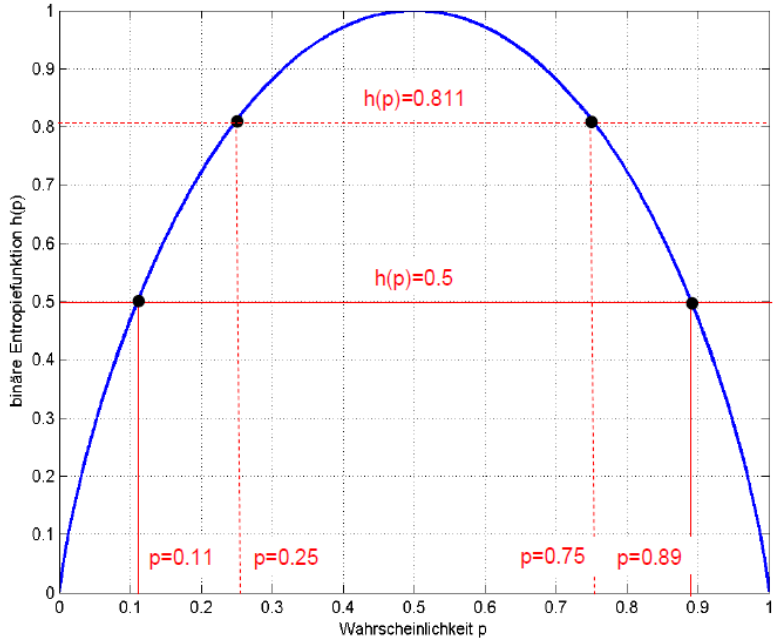
- Haupteigenschaften

- Der BSC ist **gedächtnisfrei**. Jedes vorangegangene Bit kann als „abgeschlossene Sache“ betrachtet werden. Auch falls der selten eintretende Fall auftritt, dass zum Beispiel zehn (oder eine beliebige andere Anzahl) aufeinander folgende Einsen vorliegen, ist die Wahrscheinlichkeit für eine „1“ oder eine „0“ als nächstes Bit genau je 50 %.

- Die Übergangswahrscheinlichkeit ε ist die Wahrscheinlichkeit, dass am Ausgang eine „1“ erscheint bei einer „0“ am Eingang (und umgekehrt). Die Übergangswahrscheinlichkeit ε beschreibt demnach das **Fehlerverhalten**.

- Die **Bitfehlerrate** (BER) für den Fall, dass keine Forward Error Correction angewandt wird, beträgt **ε**.

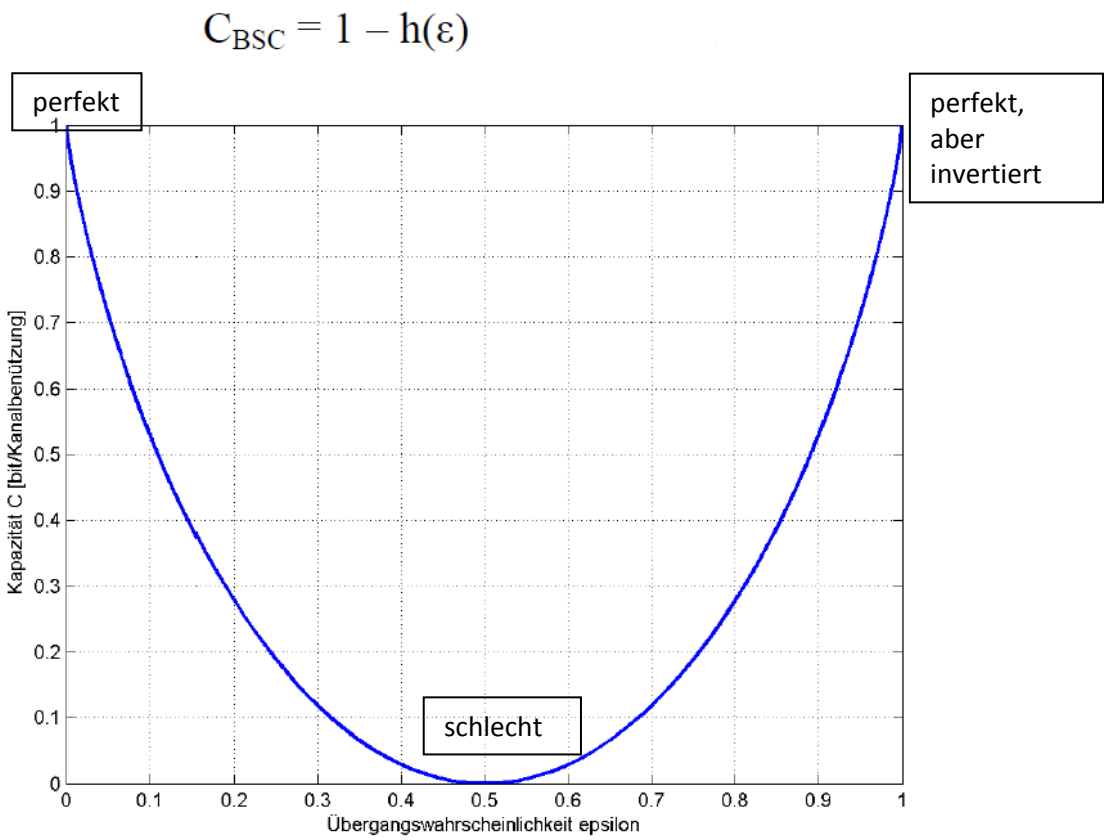
5.2.3 Die Kanalkapazität des BSC



- Binäre Entropiefunktion h(p)
- h(p) ist maximal (=1 bit), wenn die beiden Ereignisse "0" und "1" gleich wahrscheinlich sind, d.h. wenn p = 0.5
- h(p) = 0 und damit minimal, wenn p=0 oder p=1 ist
- h(p) ist konkav und grösser gleich Null im interessierenden Bereich $0 \leq p \leq 1$

- Es gilt: $H(x) = h(p) = -p * \log_2(p) + (-1) * (1-p) * \log_2(1-p)$

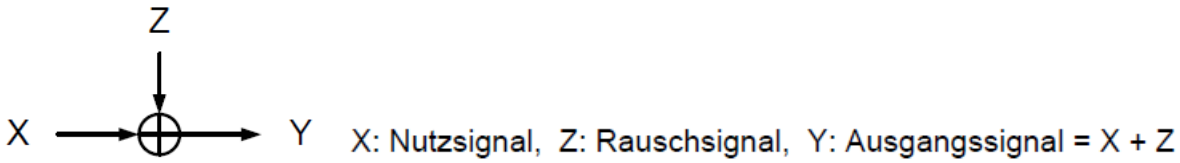
- Bei der nachfolgenden Darstellung tragen die Achsen die folgenden Werte:
  - Die x-Achse zeigt den Wert der Übergangswahrscheinlichkeit ε des BSC (Binary Symmet-ric Channel). Beträgt dieser Wert ε gleich Null, so arbeitet der BSC ideal, weil niemals ein Bitfehler stattfindet. Jede Null am Eingang wird auch als Null am Ausgang erscheinen, dasselbe gilt für eine Eins am Eingang, die ebenfalls unverfälscht übertragen wird. Beträgt ε jedoch exakt 0.5, so wird eine Eins am Eingang mit gleicher Wahrscheinlichkeit in eine Null oder eine Eins am Ausgang „verwürfelt“. Damit geht die Möglichkeit, sicher ein Bit von Eingang zum Ausgang zu übertragen, 100 %-ig verloren. Im Bereich dazwischen ist aus der Kurve ersichtlich, welche Kanalkapazität resultiert
  - Die y-Achse zeigt den Wert der Kanalkapazität in bit pro Kanalbenützung



- Rauschfreier Kanal: $\varepsilon = 0$ oder $\varepsilon = 1$ (Inversion) → C = 1 bit/Kanalbenützung
- 100%-ig verrauschter Kanal: $\varepsilon = 0.5$ → C = 0 bit/Kanalbenützung
- Ein zwischenwert an der Stelle $\varepsilon = 0.1$: Hier beträgt die Kapazität 0.53 bit/Kanalbenützung

5.2.4 Der AWGN-Kanal

- Additive White Gaussian Noise
- Additiv: Nutzsignal und Rauschsignal überlagern sich additiv



- Die Amplituden sind gaussverteilt (normalverteilt)

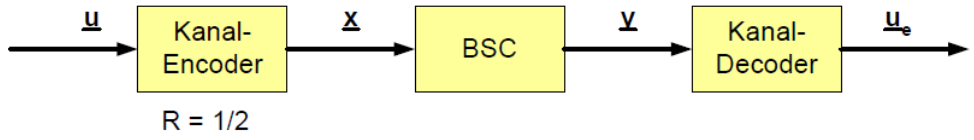
- Einschränkungen
  - Das Rauschen ist nicht weiss von Frequenzen im Berich von 0 Hz … ∞ Hz, sondern die Rauschamplitude A wird bei grösseren Frequenzen abklingen und sich asymptotisch der Nulllinie nähern
  - Das Rauschen ist möglicherweise nur angenähert gaussisch

5.2.5 Die Kanalkapazität des AWGN-Kanals

- Wiederholung Kanalkapazität C des BSC: Was bedeutet C = 0.53 bit/Kanalbenützung an der Stelle ε = 0.1? → Zuverlässige Kommunikation findet nur dann statt, falls Coderate R einen Wert von R < 0.53 besitzt

Definition der Coderate R: $\text{Coderate } R = \frac{\text{Anzahl Informationsbits}}{\text{Anzahl Codebits}}$

Kanalencoder, BSC und Kanaldecoder im Blockdiagramm dargestellt:



Kanalkapazität des AWGN-Kanals

Kanalkapazität: $C_{AWGN} = B \cdot \log_2[1 + \frac{S}{N_0 \cdot B}]$

- N_0 = Rauschleistungsdichte in Watt/Hz
- B = Bandbreite in Hz

5.2.6 Binäre Block-Codes

→ auch als (N, K) Block-Code bezeichnet

- Code, bei dem der Encoder die Informationssequenz in Blöcke aufteilt
- Jeder Block weist die Länge von **K Informationsbits** auf
- Ein Block wird als **Informationswort u** bezeichnet
- Total sind theoretisch 2^K verschiedene Informationsworte derselben Länge K möglich
- Encoder wandelt Informationswort u in ein **Codewort x** um
  - Jedes Codewort x weist die **Länge N** auf, wobei **N > K**

- Das Verhältnis $\frac{K}{N}$ ist die Coderate
- Encoder ist gedächtnislos

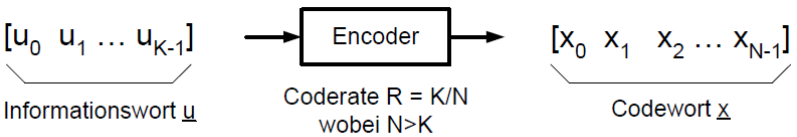


Abb. 5.9: Encoder für den (N,K) Block-Code

5.2.6.1 Begriff "systematischer (N, K) Block-Code C"

- Falls die K Informationsbits, also das Informationswort u "en bloc" auch im Codewort x erscheint, ist der Block-Code als **systematischer (N, K) Block-Code** bezeichnet. Dadurch gestaltet sich die Rücktransformation, das heisst die Umwandlung des Codewort x in das Informationswort u, relativ einfach. Die zusätzlichen (N-K) Bits, die den Informationsbits beigefügt werden, sind als "Parity Check Bits" bezeichnet. Der Begriff "Parity" hat nicht die Bedeutung des Repräsentierens einer geraden oder ungeraden Zahl, sondern der Begriff hat hier die allgemeinere obige Bedeutung

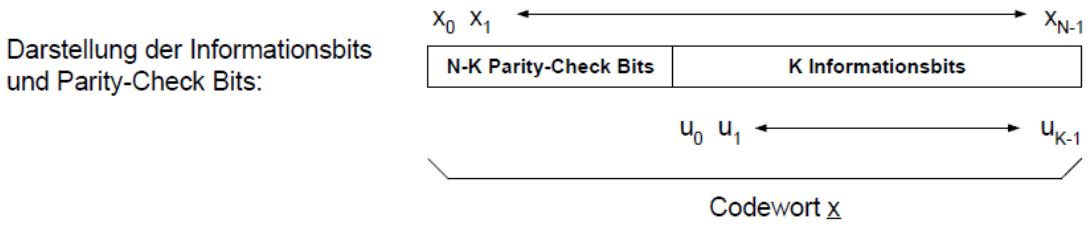


Abb. 5.10: Bildung eines Codeworts

5.2.6.2 Begriff "linearer (N, K) Block-Code C"

- Falls die **modulo-2 Summe** zweier Codewörter wieder ein Codewort ergibt, dann ist der Block Code **linear**

5.2.6.3 Begriff "linearer, zyklischer (N, K) Block-Code C"

- Falls die zyklische Verschiebung eines Codeworts wieder ein Codewort ergibt, ist der Code ausserdem zyklisch. Aufgrund dieser Eigenschaft sind die verschiedenen Codeworte sehr einfach mit Hilfe eines LFSR realisierbar

5.2.6.4 Einige Begriffe im Zusammenhang mit der Fehlerkorrektur

5.2.6.4.1 Hamming Gewicht $w_H(x)$

- Das Hamming-Gewicht $w_H(x)$ entspricht der Anzahl "1" im Codewort x

5.2.6.4.2 Hamming-Distanz $d_H(x_i, x_j)$

- Die Hamming-Distanz $d_H(x_i, x_j)$ entspricht der Anzahl unterschiedlicher Positionen in x_i *und* x_j.

Beispiel: Gegeben seien zwei Codeworte $\underline{x}_j$ und $\underline{x}_k$. Die Anzahl Bits, in denen $\underline{x}_j$ und $\underline{x}_k$ verschieden sind, ist die Hammingdistanz. Dazu ein Beispiel:

Sei $\underline{x}_j$ = 0 0 1 0 1 1

Sei $\underline{x}_k$ = 0 1 1 0 1 0

Verschieden in: x x ➔ in **zwei** Bits verschieden.

Die Hammingdistanz dieser zwei Codeworte beträgt **zwei**.

5.2.6.4.3 Minimaldistanz d_{min} des linearen (N, K) Block-Codes C

$d_{min} = min_{ij}\, d_H(x_i, x_j) = min_{ij}(w_H(x_i + x_j) = min_k w_H(x_k) = w_{min}$ wobei $(i \neq j)$

➔ Beispiel Skript Seite 12

5.2.6.4.4 Fehlerdetektion

- Gegeben sei ein Block-Code C mit der Hammingdistanz d_{min}. Dann ergibt sich:
Alle Muster mit $\leq (d_{min} - 1)$ Fehlern sind detektierbar. Total sind lediglich $2^K - 1$ Fehlermuster e undetektierbar, nämlich falls $e = x_j, \;\; y = x_i + e = x_k$
In Worten: Falls der Fehlervektor e gerade einem gültigen Codewort xj entspricht, dann ergibt die Modulo-2 Addition eines gesendeten gültigen Codeworts mit dem Fehlervektor ein anderes gültiges Codewort. Damit kann ein solcher Fehler nicht detektiert werden.
Wiederum an unserem Beispiel des (3,2) Block-Codes nachvollzogen:
Falls [0 0 0] gesendet wurde, und [0 1 1] beim Empfangsdetektor resultiert, ist der Fehlervektor e = [0 1 1].
[0 0 0] + [0 1 1] = [0 1 1] = x_k, eben wieder ein gültiges Codewort.

Ein anderer Fall: Falls [0 1 1] gesendet wurde und der Fehlervektor e = [0 1 1] beträgt, resultiert:
[0 1 1] + [0 1 1] = [0 0 0], also ebenfalls ein gültiges Codewort, wie wir wissen.

5.2.6.4.5 Fehlerkorrektur

- Das Ziel besteht darin, Fehler nicht nur zu erkennen, sondern auch zu korrigieren
- Gegeben sei wiederum ein Block-Code C mit der Hammingdistanz d_{min}. Dann gilt:

Alle Muster mit $t \leq \lfloor (d_{\min} - 1)/2 \rfloor$ Fehlern sind korrigierbar

- Bei unserem Beispiel mit den Codewörtern [0 0 0], [1 1 0], [1 0 1, [0 1 1] beträgt $d_{min} = 2$. Damit ist $\frac{d_{min}-1}{2} = 0.5$. Weil das "floor"-Zeichen bedeutet, dass der Wert auf die nächste ganze Zahl abgerundet werden muss, wird der Wert 0.5 auf null abgerundet
→ Also sin in diesem Beispiel "Muster mit null Fehlern korrigierbar", also **keine Fehler korrigierbar**!

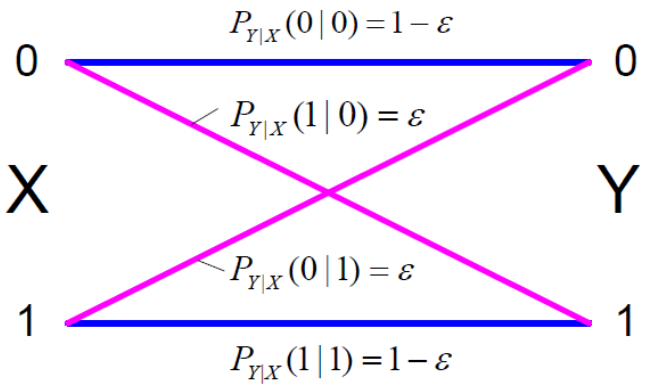
- Begriff „t-Fehlerkorrektur“: Der Begriff „t-Fehlerkorrektur“ bedeutet, dass alle Muster, die die Anzahl t oder weniger Fehler aufweisen, innerhalb der gegebenen Wahrscheinlichkeiten eines Binary Symmetric Channel korrigierbar sind. Eine Sicherheit für die Fehlerkorrektur gibt es jedoch nicht. Stets sind Wahrscheinlichkeiten die Richtgrössen.

- Für die t-Fehlerkorrektur mit einem (N,K,t)-Block-Code auf einem Binary Symmetric Channel (BSC) gilt die Beziehung (5.8). Mit anderen Worten: Einen (N,K) Block-Code, der t Fehler korri-gieren kann, bezeichnet man auch als (N,K,t) Block-Code. **Die Wahrscheinlichkeit, dass auf ei-nem BSC mit einem (N,K,t) Block-Code ein Codewort korrekt übertragen wird, ist:**

t-Fehlerkorrektur mit einem (N,K,t)-Block-Code auf einem BSC:

$$P(\underline{u} = \underline{u}_e) = \sum_{i=0}^{t} \binom{N}{i} \cdot \varepsilon^i \cdot (1-\varepsilon)^{N-i} \qquad (5.8)$$

- Darin bedeutet $u = u_e$, dass ein Fehlervektor u_e mit maximal definierter Anzahl Fehler auftreten kann.
- Der Blockcode selbst kann aber mit der Formel nicht erhalten werden. Die Formel gibt nur an, wie gross die WSK ist, dass null Fehler pro CW resultieren plus die WSK, dass 1 Fehler pro CW resultieren plus die WSK, dass zwei Fehler pro CW resultieren, usw: bis t Fehler. Falls ein (N,K,t) Block Code vorliegt, wird die Aufsummierung nach der Berechnung der WSK, dass t Fehler resultieren gestoppt, **denn so viel kann ja korrigiert werden**.



$P_{kein\ Fehler,wenn\ 1\ Bit\ gesendet} = 1-\varepsilon$

$P_{kein\ Fehler,wenn\ N\ Bit\ gesendet} = (1-\varepsilon)^N$

$P_{1\ Fehler,wenn\ 1\ Bit\ gesendet} = \varepsilon$

$P_{1\ Fehler,wenn\ 2\ Bit\ gesendet} = \varepsilon * (1-\varepsilon) + (1-\varepsilon) * \varepsilon = 2 * \varepsilon * (1-\varepsilon) = 2 * \varepsilon * (1-\varepsilon)^1$

Wie ist das zu erklären: Wenn ein Ereignis auf verschiedene Weise eintreten kann, müssen die Einzelwahrscheinlichkeiten der verschiedenen Möglichkeiten addiert werden. Der Term $\varepsilon \cdot (1-\varepsilon)$ bedeutet: Der Fehler tritt beim ersten Bit auf; das zweite Bit wird richtig übertragen.
Der zweite Term $(1-\varepsilon) \cdot \varepsilon$ bedeutet: Der Fehler tritt beim zweiten Bit auf; das erste Bit wurde richtig übertragen.

$P_{1\ Fehler,wenn\ 3\ Bits\ gesendet} = 3 * \varepsilon(1-\varepsilon)^2$

$P_{2\ Fehler,wenn\ 2\ Bits\ gesendet} = \varepsilon * \varepsilon = \varepsilon^2$

$P_{q\ Fehler,wenn\ q\ Bit\ gesendet} = \varepsilon^q$

$P_{1\ Fehler,wenn\ N\ Bit\ gesendet} = N * \varepsilon^1 * (1-\varepsilon)^{N-1}$

$$P_{\text{t Fehler wenn N Bits gesendet}} = \binom{N}{t} \cdot \varepsilon^t \cdot (1-\varepsilon)^{N-t}$$

$$\binom{N}{t} = \frac{N!}{t! \cdot (N-t)!}$$

t-Fehlerkorrektur mit einem (N,K,t)-Block-Code auf einem BSC:

$$P(\underline{u} = \underline{u}_e) = \sum_{i=0}^{t} \binom{N}{i} \cdot \varepsilon^i \cdot (1-\varepsilon)^{N-i}$$

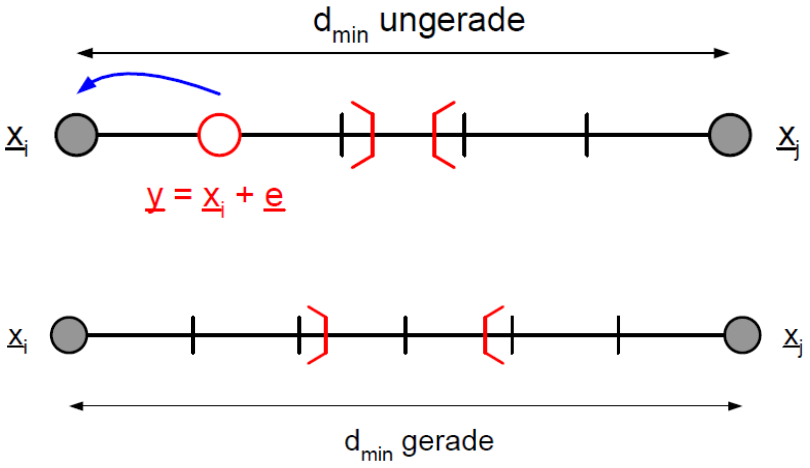
5.2.6.5 Minimum Distance Decoding

- Annahme: Wenige Fehler sind wahrscheinlicher als viele Fehler
Aufgrund dieser Annahme wird ein empfangenes Codewort, das mit einem oder evtl. mehreren Bitfehlern behaftet ist, zum „nächstgelegenen“ Codewort korrigiert

Repetition: Bei einem Block-Code C mit der Hammingdistanz d_{min} gilt gemäss (5.7) die Aussage:

Alle Muster mit $t \leq \lfloor (d_{min} - 1)/2 \rfloor$ (5.7) Fehlern sind korrigierbar

Beispiel mit minimaler Hammingdistanz von fünf:
Das gesendete Codewort sei x_i und das empfangene (fehlerbehaftete) Codewort sei y. Andere gültige Codeworte weisen mindestens 5 Bits auf, die vom Codewort x_i verschieden sein müssen. Dies gilt für alle Codeworte des so vereinbarten Codealphabets: Alle gesendeten Codeworte unterscheiden sich mindestens in fünf Bits. Zwei Codeworte, ein Codewort x_i und ein Codewort x_j haben also eine minimale Hammingdistanz von 5.
Das empfangene fehlerbehaftete CW y wird also im Empfänger korrigiert, indem die Annahme getroffen wird, das CW x_i sei das ursprünglich gesendete CW. Es wird zum „nächstgelegenen“ CW korrigiert.
Das folgende Bild soll dies veranschaulichen:



5.2.7 Generator -Matrix

Siehe Skript Seite 19

5.2.8 Linearer systematischer Block Code

Siehe Skript Seite 19

5.2.9 Parity-Check-Matrix

Siehe Skript Seite 20 bzw. Folien