Information

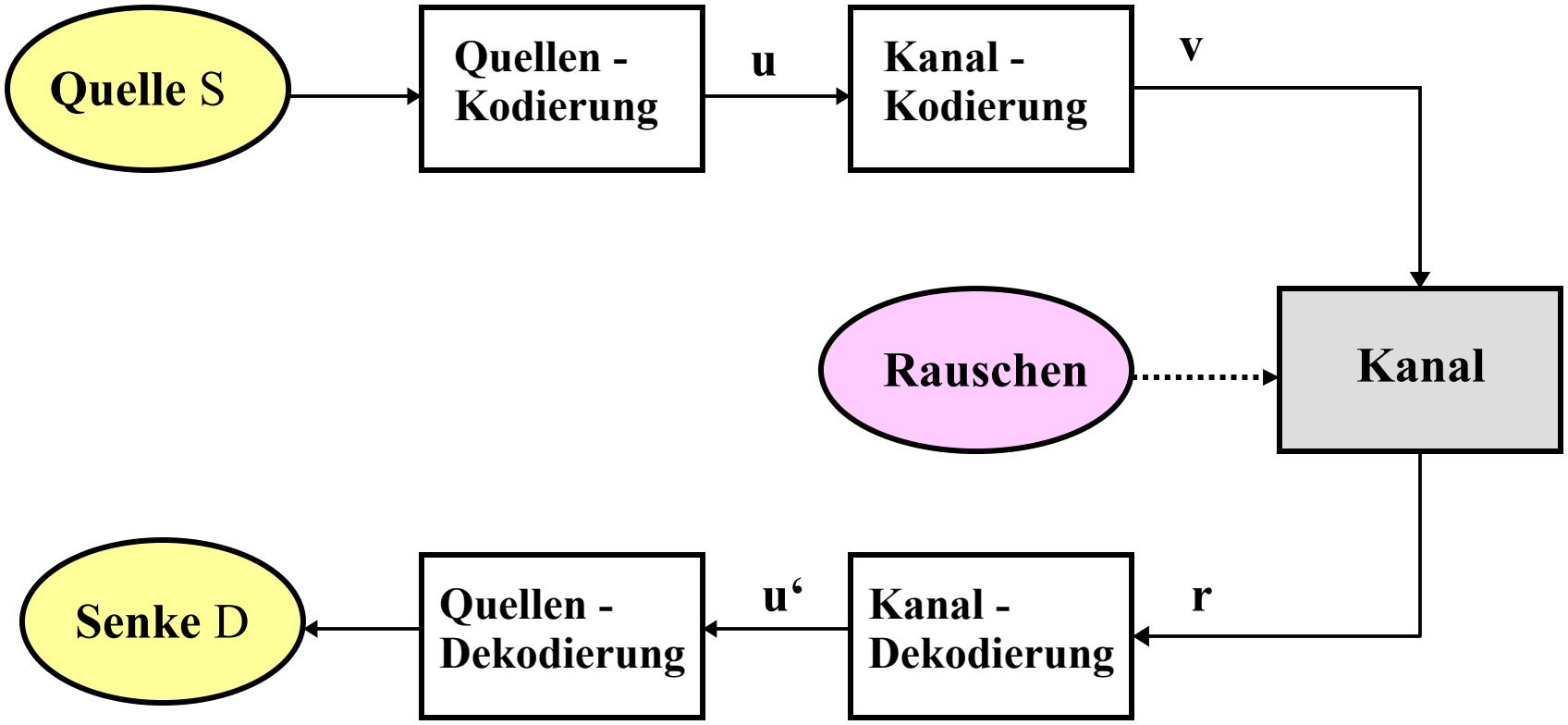
Informations- und Kodierungstheorie

Ziel: Sichere und effiziente Speicherung und Übertragung von Informationen:

- Gesendete Information soll unverfälscht beim Empfänger ankommen (räumlich, „von A nach B“).
- Gespeicherte Information soll unverfälscht zurück gewonnen werden (zeitlich, „von jetzt nach später“).

Informationsübertragung

Standardmodell eines Informationsübertragungssystems:



Informationsübertragung

Es gibt zwei grundlegend verschiedene Arten der Kodierung:

- Bei der **Quellenkodierung** wird versucht, die Nachricht möglichst kurz, d.h. ohne Redundanz, darzustellen.
Ergebnis der Quellenkodierung ist eine Nachricht **u**.

- Bei der **Kanalkodierung** werden der Nachricht **u** redundante Bits (**Prüfbits**, *control bits*, *check bits*) hinzugefügt, um eine Fehlererkennung und Fehlerkorrektur zu ermöglichen.
Ergebnis der Kanalkodierung ist ein Codewort **v**.

Auf der anderen Seite des Kanals rekonstruiert der Kanaldekodierer aus dem empfangenen Codewort **r** die Nachricht **u‘**.

Ziel: **u‘** = **u** erreichen, auch falls **r** ≠ **v**.

Information

Information is a difference that makes a difference.

Information (C. Shannon, 1948)

Umgangssprachlich ist Information eine **Auskunft** oder eine **Nachricht** oder eine **Neuigkeit** – etwas, was eine **Bedeutung** hat und unser **Wissen** steigern kann.

In der Informationstheorie hat „Information“ nichts mit Bedeutung (Semantik) zu tun, sondern nur mit der **statistischen Verteilung von Symbolen**.

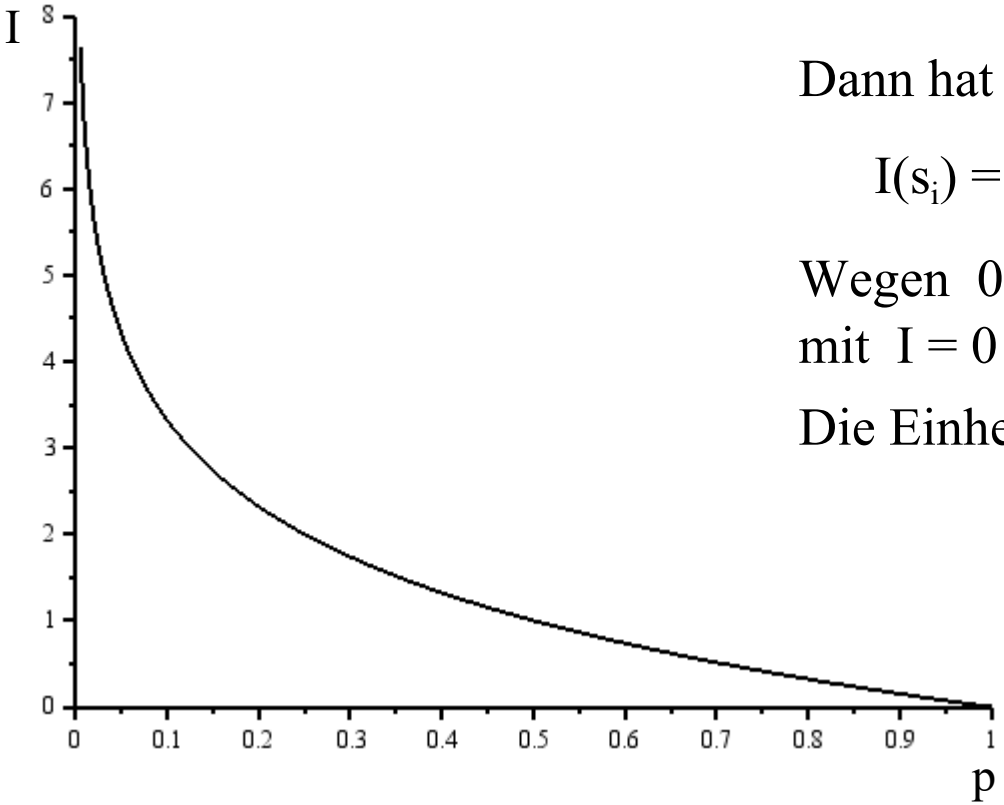
Die Bezeichnung „Informationsmenge“ statt „Information“ wäre also angebrachter, hat sich aber nicht durchgesetzt.

"*All forms of life require mechanisms to interpret information so that they can respond appropriately* (should I hug it or eat it or run?)... *Animals are informavores*." (D. Christian)

Information

Gemäß der statistischen Informationstheorie ist die Information eines Symbols s_i (eines Zeichens) umso höher, je seltener es auftritt.

Präziser: Sei p_i die Wahrscheinlichkeit des Vorkommens von s_i.



Dann hat es die **Information**

$$I(s_i) = \log_2 (1/p_i) = - \log_2 p_i$$

Wegen $0 \leq p \leq 1$ gilt $0 \leq I \leq \infty$, mit $I = 0$ für $p = 1$.

Die Einheit von $I(s_i)$ ist das **bit**.

Information

Warum wird bei der Definition der Information der Logarithmus verwendet?

Wenn zwei Ereignisse bzw. Symbole voneinander unabhängig sind, dann sollen sich die Informationen addieren, die man aus dem jeweiligen Ereignis bzw. Symbol erhält.

Aus der Wahrscheinlichkeitsrechnung wissen wir, dass sich die Wahrscheinlichkeiten p_1, p_2 von zwei unabhängigen Ereignissen zur Gesamtwahrscheinlichkeit $p = p_1 \cdot p_2$ multiplizieren lassen.

Der Logarithmus macht aus der Multiplikation eine Addition

$$\log(p_1 \cdot p_2) = \log(p_1) + \log(p_2)$$

und hat somit die gewünschte Eigenschaft.

Information

Kombinatorisch betrachtet hat eine Information,
welche n Möglichkeiten auf m Möglichkeiten reduziert,
den Informationsgehalt

$$I(\text{info}) = \log_2(n/m) \text{ bit.}$$

Bsp.: Personen erraten mit Ja/Nein-Fragen.

Annahme: Es gibt/gab 12 Milliarden Menschen
($7 \cdot 10^9$ lebendig, $5 \cdot 10^9$ tot).

Information = Reduzierung der Ungewissheit.

F: Lebt die Person noch?

A: Ja $\Rightarrow I = \log_2(12 \cdot 10^9 / 7 \cdot 10^9) = 0.7776$ bit.

Nein $\Rightarrow I = \log_2(12 \cdot 10^9 / 5 \cdot 10^9) = 1.2630$ bit.

F: Ist die Person Papst (tot oder lebendig)?

A: Ja $\Rightarrow I = \log_2(12 \cdot 10^9 / 300) = 25.2535$ bits.

Nein $\Rightarrow I = \log_2(12 \cdot 10^9 / 11.99999 \cdot 10^9) = 3.6 \cdot 10^{-8}$ bit.

Entropie

Eine **Nachricht** besteht i.d.R. aus einer Folge von Zeichen.
In der Sprache der Informationstheorie ausgedrückt:

- Eine **Quelle** S sendet **Symbole** s_i aus dem **Alphabet** Σ.
- Jedes Symbol s_i hat eine **Auftrittswahrscheinlichkeit** p_i.

Die **mittlere Information einer Quelle** ist die über die Auftrittswahrscheinlichkeiten gewichtete Summe der Einzelinformationen und wird **Entropie** H(S) genannt:

$$H(S) = \sum_{i=1}^{|\Sigma|} p_i \, I(s_i) = \sum_{i=1}^{|\Sigma|} p_i \, \log_2 \frac{1}{p_i} = -\sum_{i=1}^{|\Sigma|} p_i \, \log_2 p_i$$

Es gilt $0 \leq H(S) \leq \log_2 |\Sigma|$ sowie $\Sigma p_i = 1$ und $\lim_{p_i \to 0} p_i \cdot \log_2 \frac{1}{p_i} = 0$

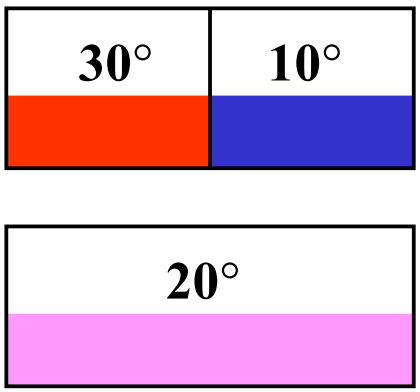
Die Einheit von H ist ebenfalls **bit**, hier manchmal auch Sh (= Shannon) genannt.

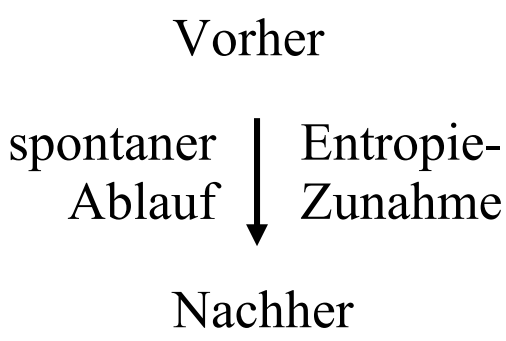
Entropie

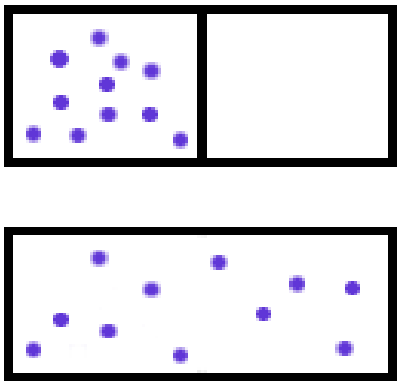
In der Physik ist die Entropie ein Maß für die Wahrscheinlichkeit („**Unordnung**“) eines abgeschlossenen Systems.

Zweiter Hauptsatz der Thermodynamik: Spontan ablaufende Prozesse verringern die Zustandswahrscheinlichkeit (Entropie) des Systems nicht.

Bsp.: Spontane Vermischung von Flüssigkeiten unterschiedlicher Temperatur oder von Gasen unterschiedlicher Konzentration.







Entropie

Saloppe Formulierung des Zweiten Hauptsatzes:
Die Unordnung nimmt immer zu.

Bsp.: 40 Karten mit 40 Namen werden auf einen Stapel gelegt.

Es gibt

- nur einen einzigen sortierten (= perfekt geordneten) Zustand
- $40!-1 \approx 8 \cdot 10^{47}$ unsortierte Zustände (Entropie ist höher).

Eine spontane Sortierung ist somit praktisch ausgeschlossen.
Eine willentliche Sortierung verringert zwar lokal die Entropie, dies wird aber an anderer Stelle mit Produktion von Wärme (= Entropiezunahme) bezahlt.

Erhöht sich die Zustandswahrscheinlichkeit und somit die Entropie des Systems, so ist der Vorgang **irreversibel** ($\Rightarrow$ Definition der **Richtung der Zeit**; Quantencomputer).

Entropie

Bsp.: Eine Quelle S über dem Alphabet

Σ ={Sieg, Niederlage, Unentschieden}

produziere die Symbolfolge

(Niederlage, Niederlage, Sieg, Niederlage, Unentschieden, Sieg, Sieg, Sieg).

Wie groß ist die Entropie der Quelle?

Dann ist $p_S = 4/8 = 0.5$

$p_U = 1/8 = 0.125$

$p_N = 3/8 = 0.375 \quad \Rightarrow$

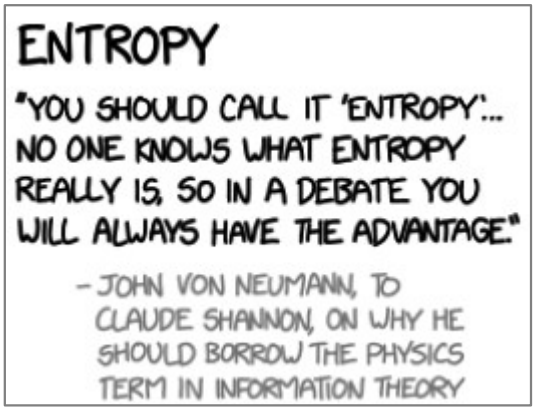
$$H(S) = 0.5 \cdot \log_2 2 + 0.125 \cdot \log_2 8 + 0.375 \cdot \log_2 8/3$$

$$= 0.5 + 0.375 + 0.5306 = 1.4056 \text{ bit}$$

Annahme:

Die beobachteten Häufigkeiten sind repräsentativ für die Wahrscheinlichkeiten.

Entropie



Keine Rolle bei der Berechnung der Entropie spielt

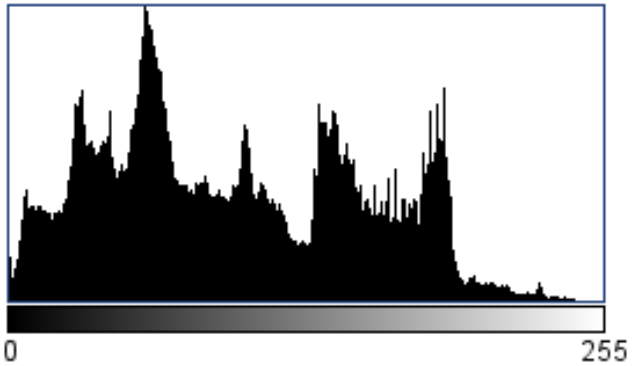
- die Reihenfolge der Symbole
- die Bedeutung der Symbole.

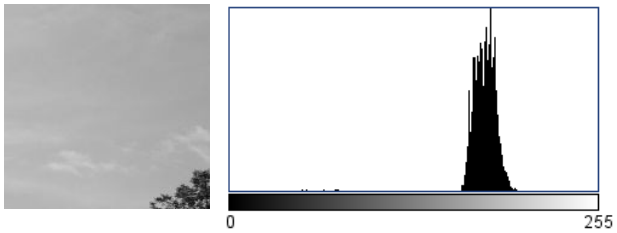
Die Symbolfolge (n, n, u, u, u, s, u, n) hat dieselbe Entropie, da sie dieselbe statistische Verteilung hat.

Ganz wichtige Annahme: Die Ereignisse (= Auftreten der Symbole) seien **statistisch unabhängig**.

Man bezeichnet die Quelle dann auch als **gedächtnislos**. Ansonsten wäre die Quelle **gedächtnisbehaftet**.

Entropie





Bsp.: Entropie von Bildern.

Die Häufigkeit der Helligkeitswerte (0..255) wird als Histogramm festgehalten.

Per Division durch die Anzahl der Pixel erhalten wir die relativen Häufigkeiten.

Für nebenstehendes Bild ist H = 7.61 bit.

Die Entropie ist maximal, wenn alle Helligkeitswerte gleich häufig vorkommen.

Die Entropie im Bildausschnitt links oben beträgt H = 5.07 bit. Das Histogramm ist weit von einer Gleichverteilung entfernt.

Entropie

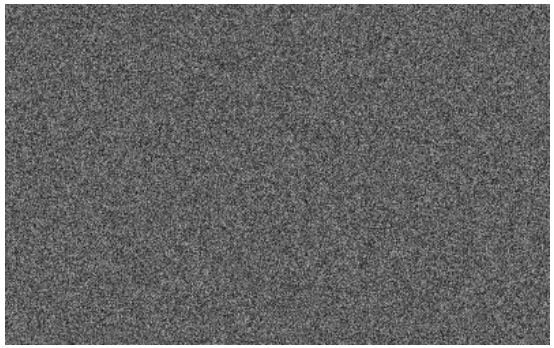
Alle nebenstehenden Bilder haben dasselbe Histogramm und somit dieselbe Entropie $H = 7.61$ bit.

Die Pixel sind lediglich umsortiert worden.

Ganz offensichtlich sind die Pixel in einem „normalen“ Bild nicht zufällig – die Welt hat ihre Naturgesetze und Regelmäßigkeiten.

Somit ist die Annahme einer statistischen Unabhängigkeit der Pixel nur selten haltbar.

Intuitiv ist die Unordnung im mittleren Bild (gedächtnisbehaftet) deutlich kleiner als im unteren Bild (gedächtnislos).



Entropie

In der Informatik (und insbesondere in der Kryptographie) interpretieren wir Entropie als die **Unsicherheit bei der Vorhersage** von Ereignissen bzw. Werten.

Bsp.: Im Vorlesungssaal geht die Tür auf. Betritt ein Mann oder eine Frau den Saal?

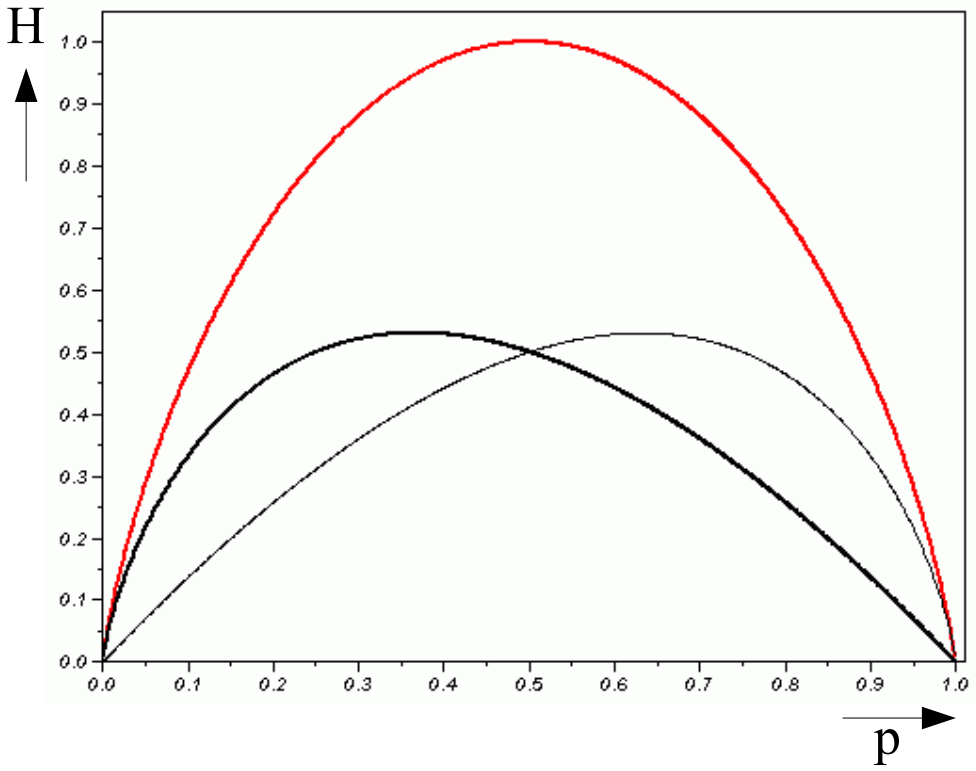
Szenario I: Informatik-Vorlesung, 90% Männer, 10% Frauen.
$\Rightarrow$ $H = 0.9 \cdot \log_2 1/0.9 + 0.1 \cdot \log_2 1/0.1 = 0.469$ bit.

Szenario II: Medizin-Vorlesung, 50% Männer, 50% Frauen.
$\Rightarrow$ $H = 0.5 \cdot \log_2 1/0.5 + 0.5 \cdot \log_2 1/0.5 = 1.0$ bit.

In Szenario I kann man also mit viel weniger Unsicherheit vorhersagen, dass ein Mann den Vorlesungssaal betreten wird.

Entropie

Für eine **binäre Quelle** S_2 gilt $|\Sigma| = 2$, mit $\Sigma = \{0, 1\}$. Wenn die Wahrscheinlichkeit von 0 gleich p ist, dann ist die von 1 gleich 1 - p.



Die Entropie $H(S_2)$ einer binären Quelle als Funktion von p ist dann

$$H(S_2) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}$$

Rot: Der Graph von $H(S_2)$.

Schwarz: Die Summanden, d.h.

$$f(p) = p \log_2 \frac{1}{p} \quad \text{sowie}$$

$$f(1-p) = (1-p) \log_2 \frac{1}{1-p}$$

Entropie

Die Extrempunkte von $H(S_2)$:

- Die Entropie erreicht ihr Maximum bei $p = 0.5 = 1-p$.

Die Unsicherheit ist also am größten, wenn beide Symbole genau gleichwahrscheinlich sind.

Bei einer Quelle mit beliebigem Alphabet wird das Maximum erreicht, wenn alle Symbole gleichwahrscheinlich sind.

- Die Entropie ist minimal, nämlich 0, bei $p = 0$ und $p = 1$.

In diesen Fällen gibt es keine Unsicherheit – die Quelle produziert nur Nullen oder nur Einsen.

Eine Quelle mit niedriger Entropie ist folglich leicht vorhersagbar (sie produziert nur wenig „Überraschungen“), während eine hohe Entropie ein Indikator für eine unsichere Vorhersagbarkeit ist.

Entropie

Wie groß ist die maximale Entropie einer Quelle?

Das Alphabet habe $|\Sigma|$ verschiedene Symbole, die gleichwahrscheinlich sind. Dann gilt für jedes Symbol $p_i = 1/|\Sigma|$, und die maximale Entropie ist

$$H_{max}(S) = \sum_{i=1}^{|\Sigma|} \frac{1}{|\Sigma|} \log_2 |\Sigma| = \log_2 |\Sigma|$$

$\Rightarrow$ Faire Münze: $H_{max}(S_2) = \log_2 2 = 1$ bit.

$\Rightarrow$ Fairer Würfel: $H_{max}(S_6) = \log_2 6 = 2.58$ bit.

Entropie

Beispiel zur Verwendung des Entropie-Konzepts: Tomašev et al. (2022) nutzen das Schachprogramm AlphaZero, um Varianten der Schachregeln zu analysieren.

Wird das Spiel durch neue Regeln diverser, d.h. stehen im Schnitt bei jedem Zug mehr sinnvolle Optionen zur Verfügung?

Zur Bewertung dieser Frage berechnen Tomašev et al. die Entropie

$$H(Ch) = -\sum_{s_{t+1}} p(s_{t+1}|s_t) \log p(s_{t+1}|s_t)$$

mit s_t = Zustand nach Zug t und p = Wahrscheinlichkeit eines Zuges gemäß den Simulationen von AlphaZero.

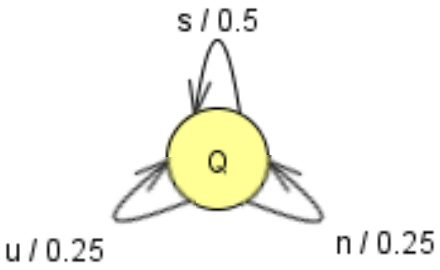
Die Diversität ist also hoch, wenn viele in etwa gleich wahrscheinliche Züge zur Auswahl stehen.

Quelle: N. Tomašev et al., *Reimagining Chess with AlphaZero*, CACM Feb. 2022.

Entropie

Gedächtnisbehaftete Quellen

Sind die Symbole einer Quelle nicht voneinander unabhängig, so sind sie leichter vorhersagbar. $\Rightarrow$ Die Entropie sinkt.



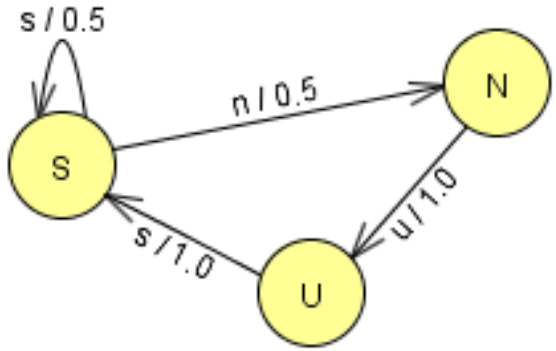
Eine gedächtnislose Quelle lässt sich darstellen als ein System mit einem einzigen Zustand, von dem Übergänge mit den Symbolwahrscheinlichkeiten auf den Zustand selbst zeigen.

Gedächtnisbehaftete Quelle hingegen haben mehrere Zustände. Die Gesamtentropie ergibt sich als gewichtete Summe aus den Entropien der jeweiligen Zustände.

Entropie

Nebenstehende gedächtnisbehaftete Quelle produziert Nachrichten wie z.B. „usnussssnussn“.

Die Symbolwahrscheinlichkeiten seien $p_s = 0.5$, $p_u = 0.25$, $p_n = 0.25$.



Die Entropie ist jedoch klein, da auf ein n immer zwingend ein u folgt und auf das u immer ein s.

Die Entropie von Zustand N ist also 0, genauso wie von U.
Im Zustand S ist die Entropie $0.5 \cdot \log_2 2 + 0.5 \cdot \log_2 2 = 1$.

Die Gesamtentropie ist somit $0.5 \cdot 1 + 0.25 \cdot 0 + 0.25 \cdot 0 = 0.5$ bits.

Entropie

Wenn eine gedächtnisbehaftete Quelle X

- sich mit Wahrscheinlichkeit p_i im Zustand P_i befindet und
- mit Wahrscheinlichkeit p_{ij} in den Zustand P_j übergeht,

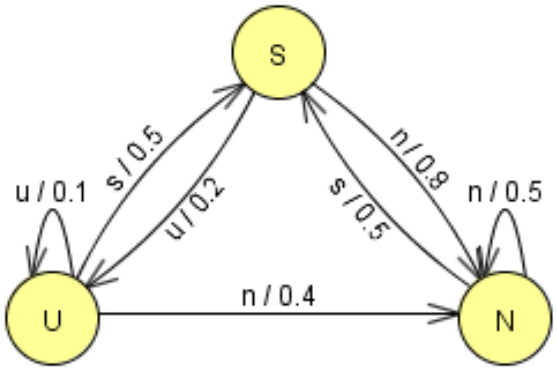
dann hat sie die Entropie

$$H(X) = \sum_{i=1}^{n} p_i \cdot H_i(X) = \sum_{i=1}^{n} p_i \cdot \Big(\sum_{j \in \mathrm{NB}(i)} p_{ij} \cdot \log_2 \frac{1}{p_{ij}} \Big)$$

Die Anzahl der Zustände n ist i.d.R. gleich der Größe des Symbolalphabets, d.h. ein Zustand pro möglichem Symbol.

NB(i) ist die Nachbarschaft von i, also die Menge aller Zustände, die im Zustandsdiagramm mit i verbunden sind.

Entropie



Im Beispiel links kann man die WS p_{ij} direkt aus dem Zustandsdiagramm ablesen.

Die WS p_s, p_n, p_u muss man berechnen, mit einer der beiden folgenden Ansätze.

Ansatz 1: Stochastische Matrix **G** aufstellen, dann $\mathbf{G}^\infty$ bestimmen.

$$\mathbf{G} = \begin{bmatrix} 0.0 & 0.8 & 0.2 \\ 0.5 & 0.5 & 0.0 \\ 0.5 & 0.4 & 0.1 \end{bmatrix} \qquad \mathbf{G}^\infty = \begin{bmatrix} 0.33333 & 0.59259 & 0.07407 \\ 0.33333 & 0.59259 & 0.07407 \\ 0.33333 & 0.59259 & 0.07407 \end{bmatrix}$$

Aus $\mathbf{G}^\infty$ lesen wir ab: $p_s = 0.33333$, $p_n = 0.59259$, $p_u = 0.07407$.

Entropie

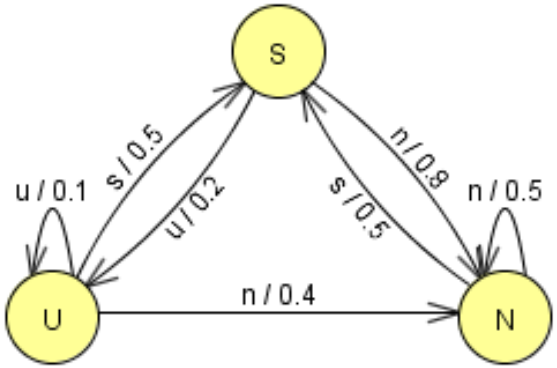
Ansatz 2: Gleichungssystem aufstellen und lösen.

$$p_s = 0.5 \cdot p_n + 0.5 \cdot p_u$$

$$p_n = 0.8 \cdot p_s + 0.5 \cdot p_n + 0.4 \cdot p_u$$

$$p_u = 0.2 \cdot p_s + 0.1 \cdot p_u$$

$$p_s + p_n + p_u = 1$$



Ohne die letzte Gleichung würde man die unzulässige Lösung $p_s = p_n = p_u = 0$ erhalten.

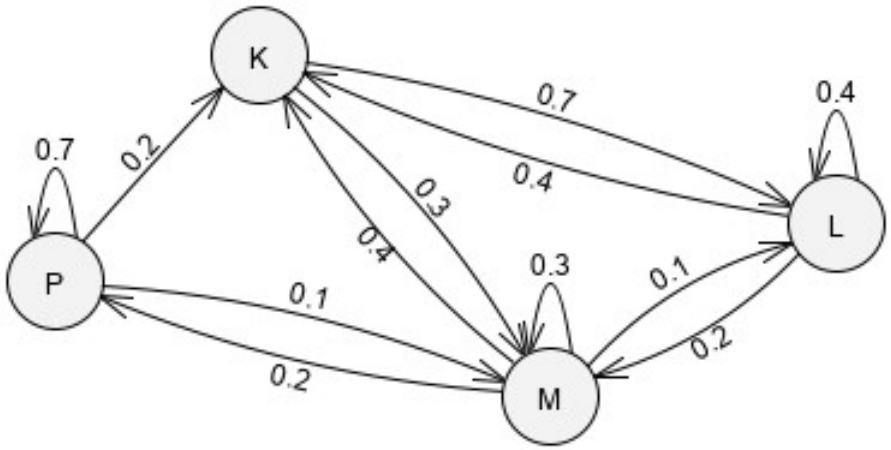
So erhält man auch hier $p_s = 0.33333$, $p_n = 0.59259$, $p_u = 0.07407$.

Diese Werte in die Formel für die Entropie eingesetzt ergibt

$$H(X) = 0.33 \cdot (0.8 \cdot \log_2 1/0.8 + \ldots) + \ldots = 0.934 \text{ bit}$$

Entropie

► *Bestimmen Sie die Entropie der folgenden Quelle X.*



H(X) = 1.3720 bit

Entropie

FAKULTÄT INFORMATIK

Abhängigkeiten zwischen Quellen / Merkmalen

Die Entropie kann eingesetzt werden, um die Nützlichkeit eines Merkmals (= Quelle) zur Klassifikation zu bestimmen.

Wenn ein Merkmal nützlich zur Klassifikation sein soll, dann muss sich durch Kenntnis des Merkmals die Entropie der zu klassifizierenden Daten verringern.

Beeinflusst das Merkmal hingegen die Klassifikation nicht, so ändert sich die Entropie nicht, das Merkmals ist nutzlos.

$\Rightarrow$ Was lernen wir durch die Quelle (= diskrete Zufallsvariable) Y hinzu, wenn wir bereits die Quelle X kennen?

Entropie

Basierend auf der Entropie definieren wir die **wechselseitige Information** (*mutual information*) $I(X;Y)$ als

$$I(X;Y) = H(X) - H(X|Y) \geq 0$$

$H(X|Y)$ ist die **bedingte Entropie** von X, wenn Y bekannt ist.

Ist $I(X;Y) = 0$, so gewinnt man durch die Kenntnis von Y nichts.

Bsp.: Ein Würfel hat die Entropie $H(X) = \log_2 6 = 2.58\,\text{bit}$.
Beobachtet man, dass die Zahl gerade ist (Y = „gerade“),
so verbleibt nur eine Entropie von $H(X|Y) = \log_2 3 = 1.58\,\text{bit}$.

Die wechselseitige Information $I(X;Y)$ ist hier somit 1 bit.

Codes

FAKULTÄT INFORMATIK

Codes und Kodierungen

Die Bedeutung der Entropie liegt u.a. darin, dass sie eine untere Schranke liefert für die Länge einer Kodierung.

Unter einem **Code** versteht man

- eine Abbildung (= Kodierung, Verschlüsselung) von einem Alphabet (Zeichenvorrat) in ein anderes oder
- das Bild einer solchen Abbildung (= Menge aller Codewörter).

ASCII-Code: Abbildung von Zeichen auf 7-bit Binärworte.

Morse-Code: Abbildung von Buchstaben auf Folgen aus den Symbolen „kurz“ und „lang“ sowie zusätzlich „Pause“.

Ein Code dient i.d.R. nicht der Verschlüsselung, sondern lediglich der Informationsdarstellung.

Codes

Codes können Codewörter von fester Länge oder von variabler Länge haben.

Längenvariable Codes erlauben eine kompaktere Kodierung, indem sie häufige Zeichen mit wenigen Bits und seltene Zeichen mit vielen Bits darstellen.

Die Dekodierung von längenvariablen Codes wird vereinfacht, wenn diese präfixfrei sind.

Ein Code ist **präfixfrei**, wenn kein Codewort Präfix eines anderen Codewortes ist.

Damit ist sicher gestellt, dass auch ohne zusätzliches Trennzeichen das Ende eines Codeworts erkannt wird.

Codes

Bsp.: Der längenvariable Code, der aus den Codeworten {abaa, abab, ba, bba} besteht, ist präfixfrei.

Der Code {abba, aba, ba, baab} ist nicht präfixfrei.

Das Telefonnummernsystem ist präfixfrei: Wenn Sie die Nummer 0815/4711 haben, dann kann es keine Nummern 0815/47110 oder 0815/4711987 etc. geben.

Der zweisymbolige Morse-Code ist nicht präfixfrei und benötigt die Pause als Trennzeichen:
„· · ·“ = „kurz-kurz-kurz“ könnte sonst z.B. gleich „EEE“ oder „EI“ oder „IE“ oder „S“ sein.

Kodierungstheorem

Quellenkodierungstheorem: Die Grenze der Komprimierbarkeit

> Für die Kodierung einer Symbolfolge der Länge n mit Entropie H benötigt man mindestens
>
> $$L_{min} = \lceil n \cdot H \rceil \text{ bits}$$

Die Differenz zwischen der tatsächlichen Kodierungslänge L und der theoretisch minimalen Kodierungslänge L_{min} wird als (Kodierungs-) **Redundanz** r_s bezeichnet:

$$r_s = L - L_{min}.$$

Die Redundanz wird oft auf ein einzelnes Zeichen bezogen, d.h.

$$r_z = L/n - H$$

mit $r_z \geq 0$ bzw. $L \geq H \cdot n$.

Kodierungstheorem

Bsp.: Die Folge S = `hallihallo` mit |S| = 10 soll kodiert werden.

Es ist p(h) = 0.2, p(a) = 0.2, p(l) = 0.4, p(i) = 0.1, p(o) = 0.1.
Die Entropie ist $H(S) = 0.2 \log_2 1/0.2 + ... + 0.1 \log_2 1/0.1 = 2.122\,\text{bit}$.

$\Rightarrow L_{min} = \lceil 10 \cdot 2.122 \rceil = 22\,\text{bit}$.

Bsp. ASCII Code: 7 bit pro Zeichen $\Rightarrow$ L = 70 bit.
Redundanz $r_s = 70 - 22 = 48\,\text{bit}$ bzw.
$r_z = 7 - 2.122 = 4.878\,\text{bit/Zeichen}$.

Bsp. Huffman-Code mit variabler Länge je Codewort:
h = 00, a = 01, l = 10, i = 110, o = 111
$\Rightarrow L = 22\,\text{bit} = L_{min}$.

Kodierungstheorem

	Deutsch	Englisch
a	6,51	8,167
b	1,89	1,492
c	3,06	2,782
d	5,08	4,253
e	17,40	12,702
f	1,66	2,228
g	3,01	2,015
h	4,76	6,094
i	7,55	6,966
j	0,27	0,153
k	1,21	0,772
l	3,44	4,025
m	2,53	2,406
n	9,78	6,749
o	2,51	7,507
p	0,79	1,929
q	0,02	0,095
r	7,00	5,987
s	7,27	6,327
t	6,15	9,056
u	4,35	2,758
v	0,67	0,978
w	1,89	2,360
x	0,03	0,150
y	0,04	1,974
z	1,13	0,074

Entropie von natürlichsprachlichen Texten

Wären alle Buchstaben gleichverteilt und statistisch unabhängig, dann ergäbe sich für das Alphabet eine Entropie $H = \log_2 26 = 4.700$ bit.

Bei der tatsächlichen Buchstabenverteilung ergäbe sich bei Gedächtnislosigkeit

- im Deutschen $H = 4.063$ bit
- im Englischen $H = 4.176$ bit.

Wegen der Groß-/Kleinschreibung und wegen der zahlreichen Sonderzeichen liegt bei Texten die Entropie höher als 4 bit. (ASCII: $\log_2 128 = 7$ bit.)

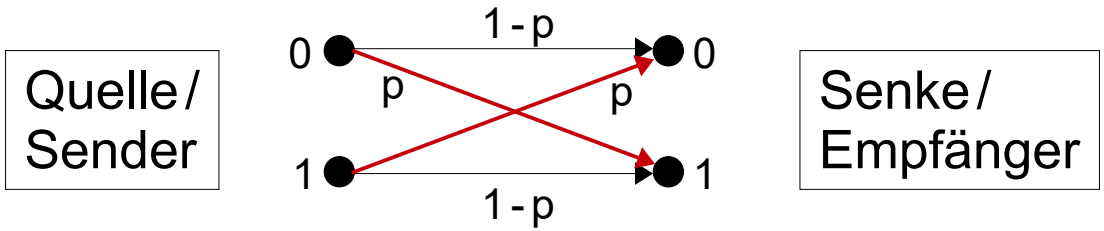
Da die Buchstabenverteilung nicht gedächtnislos ist (auf q folgt meistens u, auf sc folgt meistens h, etc.), reduziert dies die tatsächliche Entropie.

Kanalkapazität

FAKULTÄT INFORMATIK

Der Kanal: Kapazität und Rauschen

Ein **Kanal** wird charakterisiert durch seine **Kanalkapazität** C.
Diese kann man sich vorstellen als seine Bandbreite (maximale Rate), verringert um einen Teil aufgrund fehlerhafter Übertragung (Rauschen).



Der einfachste Kanal ist der hier gezeigte **binäre symmetrische Kanal** (**BSC**). Bei diesem wird ein gesendetes Bit

- mit (Fehler-) Wahrscheinlichkeit p falsch
- mit Wahrscheinlichkeit 1 - p korrekt

empfangen. Je größer p, desto niedriger ist C.

Kanalkapazität

Die Kanalkapazität C ist definiert als das Maximum (über alle möglichen Verteilungen p(S) der Quelle S) der wechselseitigen Information:

$$C = \max_{p(S)} I(S;D) = \max_{p(S)} (H(S) - H(S|D))$$

mit D = Empfänger, H = Entropie.

H(S) = H(S|D) würde bedeuten, dass eine Kenntnis der Senke (also der ankommenden Nachricht) die Entropie der Quelle nicht reduziert -- die Senke wäre vollkommen unabhängig von der Quelle, die Kanalkapazität somit 0.

H(S|D) = 0 würde bedeuten, dass sich die Quelle aus der Senke perfekt vorhersagen ließe, es also kein Rauschen gäbe.

Kanalkapazität

Die Kanalkapazität $C \geq 0$, welche sich aus den statistischen Eigenschaften der Übertragung herleiten lässt, steht in Verbindung zur Coderate.

Die **Coderate**

$$R = k/n$$

ist das Verhältnis von der Anzahl der Datenbits k zur Gesamtlänge n einer kodierten Nachricht.

Es gilt $0 < R \leq 1$. Eine kodierte Nachricht besteht also aus den eigentlichen k Datenbits, plus n-k zusätzlichen Prüfbits. Die Wahl von n und k hängt vom verwendeten Code ab.

Die Bedeutung der Kanalkapazität ist begründet im folgenden **Kanalkodierungstheorem** (**Shannons Haupttheorem,** *Shannon's noiseless coding theorem*).

Kanalkodierungstheorem

FAKULTÄT INFORMATIK

Kanalkodierungstheorem (C. Shannon, 1948):

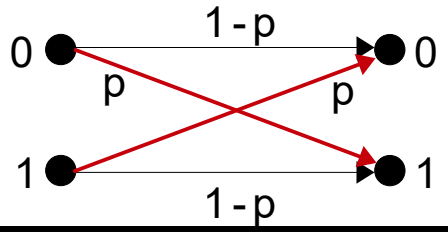
Ist die Coderate R kleiner als die Kanalkapazität C, so kann immer eine Kanalkodierung derart gefunden werden, dass Nachrichten quasi fehlerfrei (= mit beliebig kleiner Fehlerwahrscheinlichkeit) übertragen werden.

Die Kanalkapazität gibt uns also die theoretische Obergrenze für fehlerfreie Übertragung bzw. für die Anzahl der Prüfbits.

Das Theorem liefert aber kein konstruktives Verfahren zum Erreichen einer nahezu fehlerfreien Übertragung.

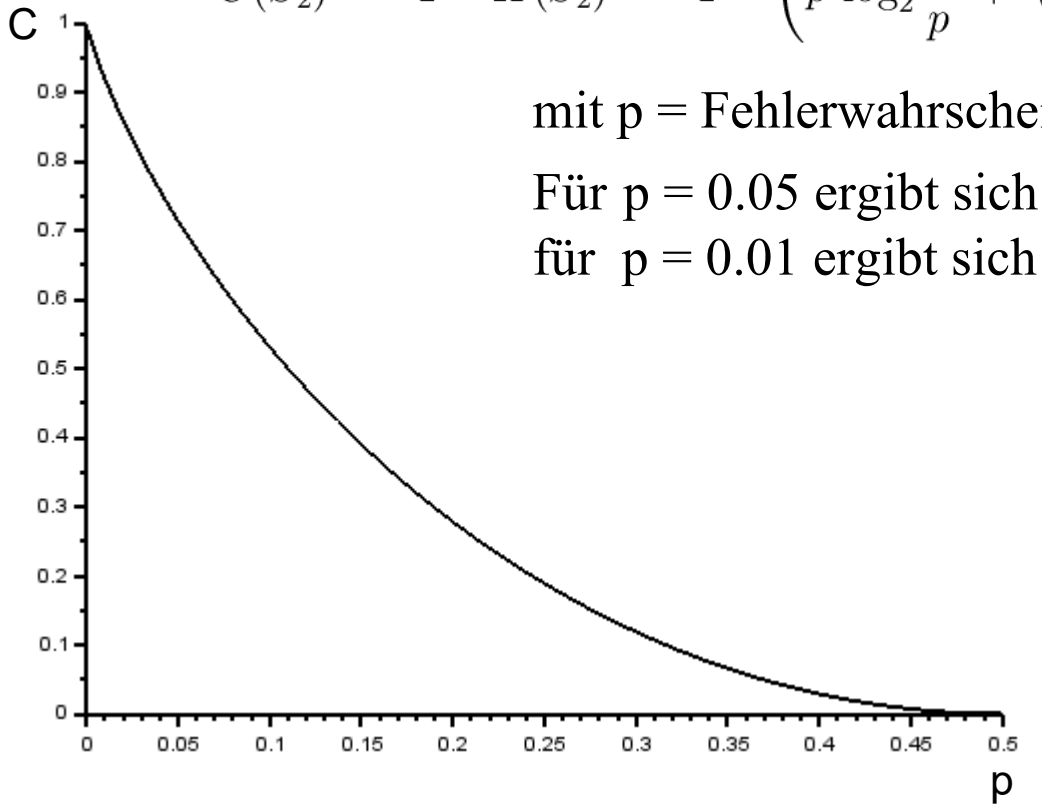
Auch kann die Kanalkodierung bei kleinem C zu sehr langen Codewörtern und damit zu ineffizienter Kodierung führen.

Kanalkodierungstheorem



Bei einem binären symmetrischen Kanal können wir die Kanalkapazität C mit einer einfachen Formel berechnen:

$$C(S_2) = 1 - H(S_2) = 1 - \left(p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p} \right)$$



mit p = Fehlerwahrscheinlichkeit eines einzelnen Bits.

Für p = 0.05 ergibt sich $C(S_2) = 0.7136$;
für p = 0.01 ergibt sich $C(S_2) = 0.9192$.

Für einen (7, 4)-Code ist
R = 4/7 = 0.5714;
für einen (15, 11)-Code ist
R = 11/15 = 0.7333.

⇒ Ein (15, 11)-Code ist für einen BSC mit p = 0.05 nicht geeignet.

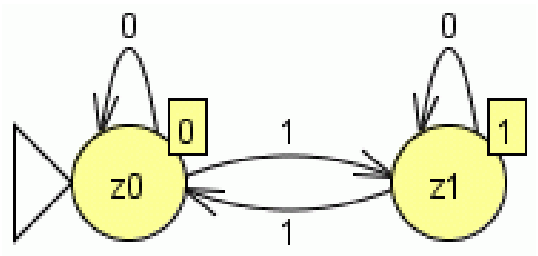
Fehlererkennung

FAKULTÄT INFORMATIK

Fehlererkennung

Die einfachste Kodierung zur Fehlererkennung besteht darin, einem Binärwort fester Länge ein **Paritätsbit** (**Prüfbit**) hinzuzufügen.

Dieses wird so berechnet, dass das **Gewicht** (= Anzahl der Einsen) immer gerade ist.



Der nebenstehende Moore-Automat kann zur laufenden Berechnung des Paritätsbits herangezogen werden: Im Zustand z_0 gibt er das Paritätsbit 0 aus, im Zustand z_1 die 1.

Das Paritätsbit kann genausogut durch die Addition der Bits des Binärwortes mittels Modulo-2-Arithmetik bestimmt werden.

Fehlererkennung

Bei der **Modulo-2-Arithmetik** verwendet man nur die Ziffern 0 und 1. Additionen und Multiplikationen werden modulo zwei ausgeführt, also

$+_2$	0	1
0	0	1
1	1	0

$\cdot_2$	0	1
0	0	0
1	0	1

- Die Modulo-2-Addition entspricht der XOR-Funktion
- Die Modulo-2-Multiplikation entspricht der UND-Funktion.
- Es gibt keinen Übertrag (d.h. einfache Berechnung in Hardware).
- Die Modulo-2-Subtraktion ist identisch zur Modulo-2-Addition.

Bsp.: Für das Wort w = 10110 errechnet sich das Paritätsbit zu

$$1 +_2 0 +_2 1 +_2 1 +_2 0 = 1$$

Das Gewicht von w ist 3 und wird durch das Paritätsbit gerade.

Fehlererkennung

Ein **Prüfziffercode** ist ein Code C,
bei dem jedes Codewort

$$w = w_1 w_2 \ldots w_n \in C, \quad \text{mit } w_i = \text{Ziffer},$$

für ein $m > 1$ und Gewichte $g_1, g_2, \ldots g_n > 0$
die folgende Gleichung erfüllt:

$$(g_1 \cdot w_1 + g_2 \cdot w_2 + \ldots + g_n \cdot w_n) \bmod m = 0$$

Binärcodes mit Paritätsbit sind ein Spezialfall des Prüfziffercodes mit $m = 2$ und $g_i = 1$.

Ein Prüfziffercode erkennt alle Einzelfehler (d.h. alle Fälle, in denen genau eine Ziffer w_i eines Codeworts nicht stimmt), wenn $ggT(g_i, m) = 1$ ist.

$\Rightarrow$ Die g_i und m müssen teilerfremd sein.

Fehlererkennung

Bedeutung der Ziffern:

1-3: Länderpräfix
(D: 400 bis 440)

4-7: Unternehmensnummer

8-12: Artikelnummer

13: Prüfziffer

Die 13 Ziffern der **EAN** (Europäische Artikelnummer = **GTIN** = *Global Trade Item Number*) erfüllen die Gleichung

$$\left(\sum_{i\%2=1} w_i \;+\; 3\cdot\sum_{i\%2=0} w_i\right) \bmod 10 = 0$$

Im nebenstehenden Beispiel gilt also

$4+1+3+5+5+8+4 + 3\cdot(0+0+5+7+2+6)$
$= 30+3\cdot 20 = 90,$ mit $90 \bmod 10 = 0.$

► *Wie lautet die fehlende Prüfziffer?*

Fehlererkennung

Für die 7 Ziffern der **PZN** (Pharmazentralnummer) gilt

$(2 \cdot w_1 + 3 \cdot w_2 + 4 \cdot w_3 + 5 \cdot w_4 + 6 \cdot w_5 + 7 \cdot w_6 + 10 \cdot w_7) \bmod 11 = 0$

PZN, bei denen die Prüfziffer $w_7 = 10$ errechnet wird, werden nicht vergeben.

Quelle: https://de.wikipedia.org/wiki/Pharmazentralnummer

Quelle: Bundesministerium des Innern

Der dt. **Personalausweis** hat 9 Stellen, mit der Prüfziffer w_{10} auf der Rückseite.

Es gilt: $(7 \cdot w_1 + 3 \cdot w_2 + 1 \cdot w_3 + 7 \cdot w_4 + 3 \cdot w_5 + 1 \cdot w_6 + 7 \cdot w_7 + 3 \cdot w_8 + 1 \cdot w_9) \bmod 10 = w_{10}$

Buchstaben werden zur Berechnung der Prüfziffer in eine Zahl umgewandelt.

Fehlerkorrektur

FAKULTÄT INFORMATIK

Fehlerkorrektur

Mit Prüfzifferncodes lassen sich Fehler (oft) erkennen, aber nicht korrigieren.

Besteht eine bidirektionale Verbindung, so kann der Empfänger den Sender zur Wiederholung der Übertragung auffordern.

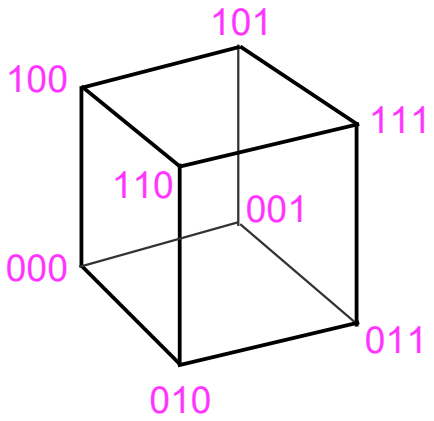
Diese Strategie reduziert den Aufwand beim Kanalkodierer, ist aber bei hohen Fehlerraten ungeeignet (vgl. Optimistische Synchronisation).

Fehlerkorrigierende Codes (*Error Correcting Codes*, **ECC**, bzw. Vorwärtsfehlerkorrektur, *Forward Error Correction*, **FEC**) verlangen zusätzliche, geschickt gewählte Bits.

Dirk Hoffmann, *Einführung in die Informations- und Codierungstheorie*, 2014. eVolltext.

Fehlerkorrektur

Prinzip der Fehlerkorrektur



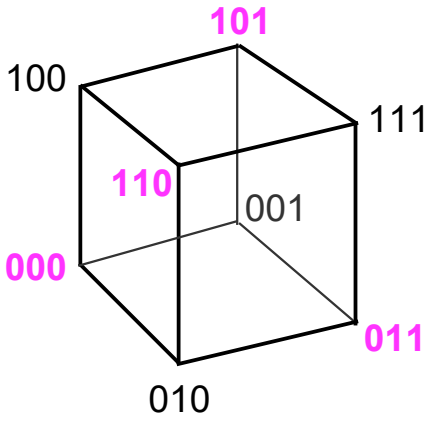
Man habe für einen Code drei Bits zur Verfügung.

Szenario 1: Man lässt acht gültige Codeworte zu.

$\Rightarrow$ Keine Möglichkeit zur Fehlererkennung, da jedes gekippte Bit wieder zu einem gültigen Codewort führt.

Die im Würfel benachbarten Codeworte unterscheiden sich in jeweils einem Bit. Jedes Codewort hat drei gültige Nachbarn.

Fehlerkorrektur



Szenario 2: Zwei Bits Nutzdaten, ein Prüfbit.

$\Rightarrow$ Vier gültige Codeworte,
z.B. 000, 011, 110, 101.

$\Rightarrow$ Ein einzelner Bitfehler kann erkannt werden:
Wenn man an diesen Codeworten genau ein Bit verändert, so kann dadurch kein anderes gültiges Codewort entstehen.

Er kann aber nicht korrigiert werden: Zum Beispiel könnte das ungültige Wort 010 gleichermaßen entstanden sein

- aus 000 durch das Umkippen des zweiten Bits oder
- aus 011 durch Umkippen des dritten Bits oder
- aus 110 durch Umkippen des ersten Bits.

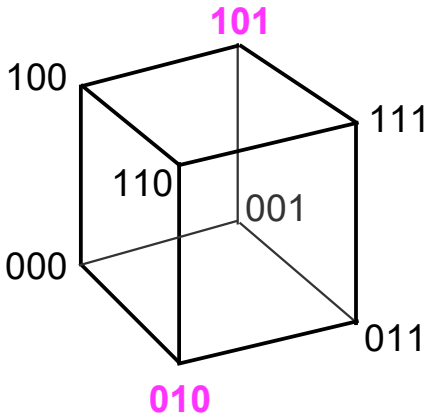
Fehlerkorrektur

Szenario 3: Ein Bit Nutzdaten, zwei Prüfbits.

$\Rightarrow$ Zwei gültige Codeworte, z.B. 010, 101.

$\Rightarrow$ Ein einzelner Bitfehler kann erkannt und korrigiert werden:
Zuordnung des ungültigen Codeworts zum nächstgelegenen zulässigen Codewort, z.B.

$110 \rightarrow 010$; $111 \rightarrow 101$.



Man hätte als Codeworte auch 000 und 111 verwenden können.

Kodierungsvorschrift: Jedes Bit dreimal hintereinander senden.

Dekodierungsvorschrift: Mehrheit in jedem Dreierblock finden.

Fehlerkorrektur

Die Anzahl der Bits, in denen sich zwei Codeworte w_1, w_2 unterscheiden, wird als **Hamming-Abstand** (oder **Hamming-Distanz**) $d_H(w_1, w_2)$ bezeichnet.

Bsp.: $d_H(01110, 00101) = 3$.

Der Hamming-Abstand ist auch gleich dem **Gewicht** w_g (= Anzahl der Einsen) der Differenz (modulo 2) der Codeworte.

Wie oben gesehen ist die Differenz gleich der Summe sowie gleich der XOR-Funktion, also

$$w_g(01110 -_2 00101) = w_g(01110 +_2 00101) = w_g(01011) = 3.$$

Fehlerkorrektur

Die **Code-Distanz** $d_{min}(C)$ eines Codes C ist gleich dem minimalen Hamming-Abstand aller Codewort-Paare von C:

$$d_{min}(C) = \min \{d_H(w_i, w_j) \mid w_i, w_j \in C \text{ und } w_i \neq w_j\}$$

Die Fähigkeit eines Codes C, Fehler zu erkennen und zu korrigieren, hängt von $d_{min}(C)$ ab:

Das **Erkennen** von r-Bit Fehlern erfordert $d_{min}(C) \geq r + 1$.

Die **Korrektur** von r-Bit Fehlern erfordert $d_{min}(C) \geq 2r + 1$.

Bsp.: Code C = {000, 011, 110, 101}.

$\Rightarrow$ $d_H(w_i, w_j) = 2$ für alle Codewort-Paare (w_i, w_j), also $d_{min}(C) = 2$.
Folglich können 1-bit Fehler erkannt werden.

Bsp.: Code C = {010, 101}.

$\Rightarrow$ $d_H(w_i, w_j) = 3 = d_{min}(C)$ für dieses Codewort-Paar.
Also können 2-bit Fehler erkannt und 1-bit Fehler korrigiert werden.

Fehlerkorrektur

Zahlreiche Fragen drängen sich auf:

- Wie entwirft man Codes C mit großer Code-Distanz $d_{min}(C)$?
- Wie kann man mit C effizient eine Nachricht kodieren?
- Wie kann man eine kodierte Nachricht effizient dekodieren?
- Wie optimiert man Coderate und Restfehlerwahrscheinlichkeit?

Die **Restfehlerwahrscheinlichkeit** ist die Wahrscheinlichkeit, dass ein Fehler trotz ECC nicht korrigiert wird.

Obacht: Bei ECCs kann es sehr wohl vorkommen, dass ein Fehler als solcher zwar erkannt, aber “falsch korrigiert” wird.

Fehlerkorrektur

Grundsätzliche Einordnung von Codes zur Kanalkodierung:

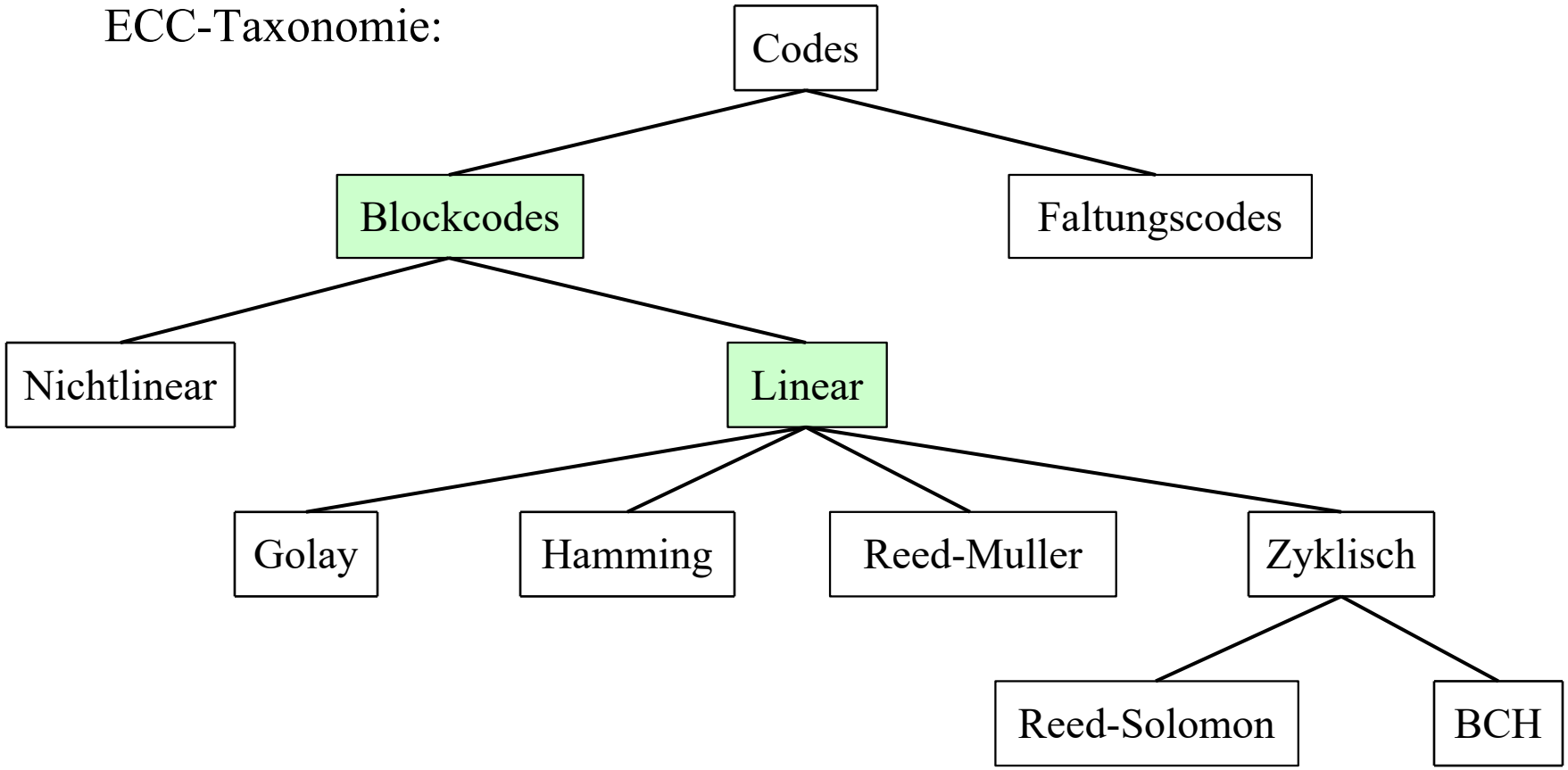
- **Blockcodes**. Bei der Blockkodierung wird die zu kodierende Nachricht in gleichgroße Blöcke unterteilt. Jeder dieser Blöcke wird um eine feste Anzahl Prüfbits erweitert.

- **Faltungscodes**. Auch Faltungskodierer teilen die Nachricht in Blöcke auf. Bei der Kodierung werden jedoch nicht nur der aktuelle, sondern auch einige vorhergehende Blöcke berücksichtigt.

Die folgende Taxonomie klassifiziert einige weit verbreitete Codes.

Fehlerkorrektur

ECC-Taxonomie:



Fehlerkorrektur

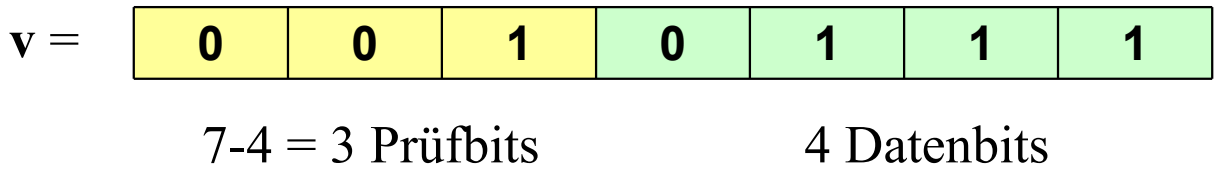
Ein **Blockcode** unterteilt die binäre zu kodierende Nachricht in Blöcke gleichbleibender Länge. Jeder Block **u** hat die Länge k.

Der Kodierer transformiert **u** in ein binäres Codewort **v** der Länge n, mit n > k. Ein solcher Code wird als **(n, k)-Code** bezeichnet.

Ein Codewort **v** der Länge n besteht also aus k **Datenbits** („Nutzbits“) und m = n-k **Prüfbits**.

Bsp.: Eine Nachricht **u** = (0111) könnte mit einem (7, 4)-Code als Codewort **v** = (0 0 1 0 1 1 1) kodiert werden.

Die Coderate des (7, 4)-Codes ist R = k/n = 4/7 = 0.5714.



Fehlerkorrektur

Ein **linearer Blockcode** ist ein Blockcode, bei dem die Summe zweier Codewörter wieder ein Codewort ist.

Ein linearer Blockcode ist also abgeschlossen unter $+_2$.

Bsp.: Zum linearen (7, 4)-Code gehören die Codewörter

$w_1 = (0\ 1\ 0\ 1\ 1\ 1\ 0)$ und
$w_2 = (0\ 0\ 1\ 0\ 1\ 1\ 1)$. Dann ist auch
$w_1 +_2 w_2 = (0\ 1\ 1\ 1\ 0\ 0\ 1)$ ein gültiges Codewort.

Ein Codewort ist nichts anderes als ein **Vektor** bzw. n-Tupel, für welches die Gesetze der **linearen Algebra** gelten.

Fehlerkorrektur

Insbesondere kann man für einen linearen (n, k)-Blockcode k linear unabhängige Codewörter $\mathbf{g}_i$ finden, so dass sich alle anderen Codewörter als **Linearkombination** dieser k Codewörter darstellen lassen.

Jene k Codewörter bilden folglich eine **Basis** für einen Unterraum des n-dimensionalen Vektorraums.

Bsp.: Beim (7, 4)-Code gibt es insgesamt $2^4 = 16$ verschiedene Codewörter. Von diesen lassen sich vier auswählen, welche die Basis für einen 4-d Unterraum des 7-d Vektorraums bilden.

Jedes Codewort **v** eines linearen (n, k)-Blockcodes ist mit geeigneten Koeffizienten $a_0 \ldots a_{k-1}$ somit darstellbar als Linearkombination

$$\mathbf{v} = a_0 \mathbf{g}_0 + a_1 \mathbf{g}_1 + \ldots + a_{k-1} \mathbf{g}_{k-1}$$

Fehlerkorrektur

Ein Nachrichtenblock $\mathbf{u} = (u_0\ u_1\ \ldots\ u_{k-1})$ wird dann als Codewort **v** kodiert, indem wir obige Koeffizienten a_i gleich u_i setzen:

$$\mathbf{v} = u_0\,\mathbf{g}_0 + u_1\,\mathbf{g}_1 + \ldots + u_{k-1}\,\mathbf{g}_{k-1}$$

Zu diesem Zeitpunkt wissen wir nur, dass in linearen Blockcodes geeignete Basisvektoren $\mathbf{g}_i$ existieren.

Wie diese konkret aussehen können, werden wir gleich für den (7, 4)-Code sehen. Eine systematische Herleitung der $\mathbf{g}_i$ betrachten wir erst weiter unten anhand von Hamming-Codes.

Die Codewörter **u**, **v**, $\mathbf{g}_i$ usw. sind Zeilenvektoren.

Fehlerkorrektur

Es ist also

$$\mathbf{v} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \cdots + u_{k-1}\mathbf{g}_{k-1} = (u_0, u_1, \ldots u_{k-1}) \cdot \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \mathbf{u} \cdot \mathbf{G}$$

mit

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix}$$

Die k×n Matrix G heißt **Generatormatrix** des linearen (n, k)-Blockcodes.

Fehlerkorrektur

Eine Generatormatrix (von mehreren möglichen) für einen (7, 4)-Code ist

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

G erzeugt die nebenstehenden Codeworte. Zum Beispiel ist das Codewort für die Nachricht $\mathbf{u} = [0\ 0\ 1\ 1]$ gleich

$$\mathbf{v} = \mathbf{u} \cdot \mathbf{G} = (0\ 1\ 0\ 0\ 0\ 1\ 1) = \mathbf{g}_2 + \mathbf{g}_3$$

► *Überzeugen Sie sich davon, dass nebenstehender Code ein linearer Blockcode ist.*

u	v
0000	0000000
1000	1101000
0100	0110100
1100	1011100
0010	1110010
1010	0011010
0110	1000110
1110	0101110
0001	1010001
1001	0111001
0101	1100101
1101	0001101
0011	0100011
1011	1001011
0111	0010111
1111	1111111

Fehlerkorrektur

Die Generatormatrix **G** oben hat die Form

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} = [\, \mathbf{P}\ \mathbf{I}_k \,]$$

G setzt sich in diesem Fall zusammen aus einer k × (n-k) Matrix **P** und der k × k Einheitsmatrix $\mathbf{I}_k$.

Wegen $\mathbf{I}_k$ sind die letzten k Bits von **v** identisch mit den k Bits der Nachricht **u**, während die ersten n-k Bits von **v** die Prüfbits sind.

Ein Code mit dieser Eigenschaft heißt **systematischer linearer Blockcode**. Ein linearer Blockcode lässt sich immer in eine äquivalente systematische Form bringen.

Fehlerkorrektur

Zur Fehlerkorrektur benötigen wir die **Prüfmatrix H**, die direkt aus der Generatormatrix **G** abgeleitet wird:

Ist $\mathbf{G} = [\,\mathbf{P}\ \mathbf{I}_k\,]$, dann ist $\mathbf{H} = [\,\mathbf{I}_{n-k}\ \mathbf{P}^T]$.

Die (n-k)×n Prüfmatrix **H** ist also nichts anderes als eine Umordnung von **G**, sie enthält folglich dieselbe Information.

Bsp.: Sei wie oben

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Dann ist} \quad \mathbf{P} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

und somit $\mathbf{P^T} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$ ⇨ $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$

Fehlerkorrektur

Für Generatormatrix **G** und Prüfmatrix **H** gilt immer: $\mathbf{G} \cdot \mathbf{H}^T = \mathbf{0}$.

Bsp.: Seien **G** und **H** Generator- und Prüfmatrix für einen systematischen (7, 4)-Code. Dann gilt:

$$\mathbf{G} \cdot \mathbf{H}^{\mathbf{T}} = \begin{bmatrix} p_{00} & p_{01} & p_{02} & 1 & 0 & 0 & 0 \\ p_{10} & p_{11} & p_{12} & 0 & 1 & 0 & 0 \\ p_{20} & p_{21} & p_{22} & 0 & 0 & 1 & 0 \\ p_{30} & p_{31} & p_{32} & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \\ p_{30} & p_{31} & p_{32} \end{bmatrix}$$

Multipliziert man eine Zeile $\mathbf{g}_i$ mit einer Spalte $\mathbf{h}_j^T$, so gilt immer

$$\mathbf{g}_i \cdot \mathbf{h}_j^T = 2 \cdot \mathbf{p}_{ij} = 0 \qquad \text{für alle i, j,}$$

denn in Modulo-2-Arithmetik ist $2 \cdot_2 0 = 0 = 2 \cdot_2 1$.

Fehlerkorrektur

$\mathbf{G} \cdot \mathbf{H}^T = \mathbf{0}$ heißt, dass jede Zeile von $\mathbf{G}$ orthogonal zu jeder Zeile aus $\mathbf{H}$ ist (zwei Vektoren sind senkrecht zueinander, wenn ihr Skalarprodukt gleich Null ist).

Daraus folgt:

- $\mathbf{v}$ ist genau dann ein gültiges Codewort, wenn $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$ gilt:

$$\mathbf{v} \cdot \mathbf{H}^T = \mathbf{u} \cdot \mathbf{G} \cdot \mathbf{H}^T = \mathbf{u} \cdot \mathbf{0} = \mathbf{0}$$

- $\mathbf{v} \cdot \mathbf{H}^T \neq \mathbf{0}$ bedeutet, dass $\mathbf{v}$ kein gültiges Codewort sein kann.

Wollen wir also testen, ob ein empfangener Nachrichtenblock $\mathbf{r}$ ein gültiges Codewort ist, so multiplizieren wir $\mathbf{r}$ mit $\mathbf{H}^T$ und schauen nach, ob das Ergebnis ein Nullvektor ist.

Fehlerkorrektur

Zur Fehlerkorrektur berechnet man mit Hilfe von $\mathbf{H}^T$ aus der Nachricht **r** das **Syndrom s** der Länge m = n - k:

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (s_0, s_1, \ldots s_{n-k-1})$$

Bsp.: Sei $\mathbf{r} = (0\ 1\ 0\ 0\ 0\ 1\ 0)$ und $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$

Dann ist $\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (1\ 0\ 1)$. Aus $\mathbf{s} \neq \mathbf{0}$ folgt **r** = fehlerhaft.

Ein empfangener Nachrichtenblock **r** ist die Summe aus einem gültigen Codewort **v** und einem **Fehlervektor e**, also

$$\mathbf{r} = \mathbf{v} +_2 \mathbf{e} \quad \text{bzw.} \quad \mathbf{e} = \mathbf{r} -_2 \mathbf{v} = \mathbf{r} +_2 \mathbf{v}$$

Bsp.: $\mathbf{e} = (0\ 1\ 0\ 0\ 0\ 0\ 1) \Rightarrow$ Bits 2 und 7 sind fehlerhaft.

$\mathbf{e} = (0\ 0\ 0\ 0\ 0\ 0\ 0) \Rightarrow$ kein Bitfehler.

Fehlerkorrektur

Wenn nur ein einziges Bit fehlerhaft ist, so kann man aus dem Syndrom die Position des fehlerhaften Bits ableiten und somit korrigieren.

Weil das Syndrom eines gültigen Codeworts gleich **0** ist, hängt das Syndrom nicht vom Codewort **v** ab, sondern ausschließlich von **e** :

$$\begin{aligned} \mathbf{s} &= \mathbf{r} \cdot \mathbf{H}^T = (\mathbf{v} +_2 \mathbf{e}) \cdot \mathbf{H}^T \\ &= (\mathbf{v} \cdot \mathbf{H}^T) +_2 (\mathbf{e} \cdot \mathbf{H}^T) \\ &= \mathbf{0} +_2 (\mathbf{e} \cdot \mathbf{H}^T) \end{aligned}$$

Bsp.: Sei $\mathbf{v} = (0\ 1\ 0\ 0\ 0\ 1\ 1)$ und $\mathbf{e} = (0\ 0\ 0\ 0\ 0\ 0\ 1)$, also $\mathbf{r} = \mathbf{v} +_2 \mathbf{e} = (0\ 1\ 0\ 0\ 0\ 1\ 0)$. Wie gesehen ist dann $\mathbf{s} = (1\ 0\ 1)$.

Sei hingegen $\mathbf{v} = (1\ 0\ 0\ 0\ 1\ 1\ 0)$ und $\mathbf{e} = (0\ 0\ 0\ 0\ 0\ 0\ 1)$ unverändert. Dann ist $\mathbf{r} = (1\ 0\ 0\ 0\ 1\ 1\ 1)$ und $\mathbf{s} = (1\ 0\ 1)$ ebenfalls unverändert.

Ein Bitfehler in der letzten Position verursacht also beim gegebenen (7, 4)-Code immer das Syndrom $\mathbf{s} = (1\ 0\ 1)$.

Fehlerkorrektur

Die Dekodierungsstrategie sieht dann so aus:

Wir berechnen vorab für verschiedene Fehlervektoren **e´** das zugehörige Syndrom **s** und legen es in einer Tabelle, der **Syndromtabelle**, ab. Diese ist eine 1:1-Zuordnung, d.h. $\mathbf{s} \Leftrightarrow \mathbf{e´}$.

Für jeden empfangenen Nachrichtenblock **r** berechnet der Dekodierer dessen Syndrom **s**.

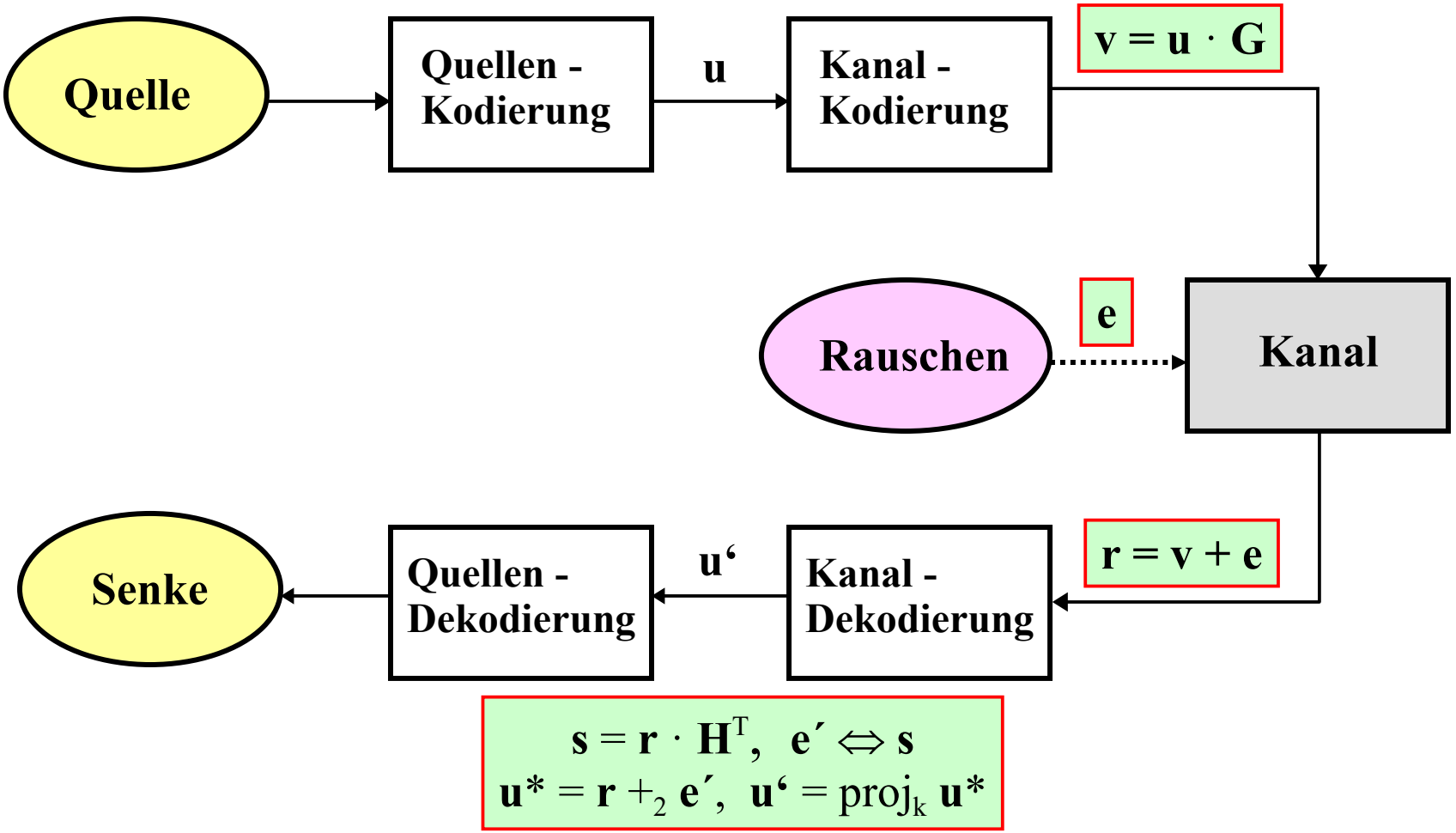
Ist $\mathbf{s} \neq \mathbf{0}$, so holen wir den zugehörigen Fehlervektor $\mathbf{e´} \Leftrightarrow \mathbf{s}$ aus der Syndromtabelle und addieren ihn zu **r**, also $\mathbf{u^*} = \mathbf{r} +_2 \mathbf{e´}$.

Die in **u*** noch enthaltenen m = n-k Prüfbits, werden vom Dekodierer entfernt. Wir erhalten $\mathbf{u´} = \text{proj}_k\, \mathbf{u^*}$ = die k Datenbits von **u***.

Idealerweise ist dann $\mathbf{u´} = \mathbf{u}$. Dieser „Idealfall“ heißt beim (7, 4)-Code, dass höchstens ein einziger Bitfehler aufgetreten ist, ansonsten kommt es zu einer fehlerhaften Dekodierung.

Fehlerkorrektur

Kodierung/Dekodierung mit linearen Blockcodes:



Fehlerkorrektur

Wie oben gesehen bestimmt die Code-Distanz $d_{min}(C)$ die Korrekturfähigkeit eines Codes C.

Für lineare Blockcodes gilt der folgende Satz:

> Die Code-Distanz $d_{min}(C)$ des Codes C ist gleich dem minimalen Gewicht $w_g(\mathbf{v})$ aller Codeworte $\mathbf{v} \in C$ (außer dem Nullwort).

Dieser Satz folgt unmittelbar aus der Tatsache, dass die Differenz (= Summe) zweier Codeworte wieder ein Codewort ist.

Beim (7, 4)-Code brauchen wir also nur die Gewichte von $2^4-1 = 15$ Codeworten zu berechnen anstelle von $16 \cdot 15/2 = 120$ Differenzen.

Für den (7, 4)-Code entnimmt man der Codewort-Tabelle $d_{min}(C) = 3$.

Fehlerkorrektur

Die Code-Distanz $d_{min}(C)$ eines (n, k)-Codes lässt sich nach oben begrenzen gemäß dem **Satz von Singleton** (1964):

$$d_{min}(C) \leq n - k + 1$$

Bsp.: Für einen Paritätsbitcode der Länge n hat man k = n-1 Datenbits und $d_{min}(C) = 2$.

Bsp.: Für den (n, 1)-Wiederholungscode (d.h. jedes Zeichen wird n-mal wiederholt) hat man k = 1 und $d_{min}(C) = n$.

Bsp.: Für die nachfolgenden Hamming-Codes gilt immer $d_{min}(C) = 3$; die Werte für (n-k+1) sind hingegen 4, 5, 6, ... für steigende Werte von m.

Fehlerkorrektur

Hamming-Codes (R. Hamming, 1950)

Ein **Hamming-Code** ist ein linearer (n, k)-Blockcode, bei dem in Abhängigkeit von der Anzahl $m \geq 3$ der Prüfbits nur die folgenden Werte für n und k zulässig sind:

$$n = 2^m - 1 \qquad k = n - m = 2^m - m - 1$$

m	n	k	R = k/n
3	7	4	0.571
4	15	11	0.733
5	31	26	0.839
6	63	57	0.905
7	127	120	0.945

Mit wachsendem m verbessert sich die Coderate R.

Aber: Je länger das Codewort, desto größer das Risiko, dass ein Block zwei oder mehr Fehler aufweist und somit nicht mehr korrigierbar ist.

$\Rightarrow$ Hamming-Codes mit $m > 4$ eignen sich nur für Kanäle mit niedrigem Rauschen.

Fehlerkorrektur

Wie gesehen hat die Prüfmatrix **H** generell die Form $\mathbf{H} = [\mathbf{I}_{n-k}\ \mathbf{P}^T]$, wobei **P** aus der Generatormatrix $\mathbf{G} = [\ \mathbf{P}\ \mathbf{I}_k\ ]$ ableitbar ist.

Bei Hamming-Codes können wir **H** sogar direkt hinschreiben: Neben allen Spalten mit dem Gewicht 1, welche $\mathbf{I}_{n-k}$ ausmachen, besteht **H** aus allen Vektoren mit dem Gewicht größer als 1, welche dann $\mathbf{P}^T$ bilden. Aus $\mathbf{P}^T$ lässt sich dann **G** ableiten.

Bsp.: Für den (7, 4)-Code ist eine mögliche Prüfmatrix **H**

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad \text{mit} \quad \mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Das Vertauschen von Spalten von **H** bewirkt lediglich eine Vertauschung der Codeworte.

Fehlerkorrektur

Bsp.: Die Bitsequenz $\mathbf{b} = (1011011011010001)$ soll mit einem (7, 4)-Code kanalkodiert werden.

Die kodierten Blöcke werden wir willkürlich durch zufällige Fehlervektoren verändern.

Die verrauschten Nachrichtenblöcke werden dekodiert und mit der Original-Bitsequenz verglichen.

Wir verwenden **H** und **G** von der vorherigen Folie.

Wir unterteilen **b** in vier Blöcke mit je k = 4 Nutzbits, also $\mathbf{b} = (\mathbf{u}_0\ \mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3)$ mit $\mathbf{u}_0 = (1011)$

$\mathbf{u}_1 = (0110)$

$\mathbf{u}_2 = (1101)$

$\mathbf{u}_3 = (0001)$.

Fehlerkorrektur

Die gesendeten Codeworte sind dann

$$\mathbf{v}_0 = \mathbf{u}_0 \cdot \mathbf{G} = (0\ 1\ 0\ 1\ 0\ 1\ 1)$$

$$\mathbf{v}_1 = \mathbf{u}_1 \cdot \mathbf{G} = (1\ 1\ 0\ 0\ 1\ 1\ 0)$$

$$\mathbf{v}_2 = \mathbf{u}_2 \cdot \mathbf{G} = (1\ 0\ 0\ 1\ 1\ 0\ 1)$$

$$\mathbf{v}_3 = \mathbf{u}_3 \cdot \mathbf{G} = (1\ 1\ 1\ 0\ 0\ 0\ 1).$$

Als Fehlervektoren wählen wir willkürlich $\mathbf{e}_0 = (0\ 0\ 0\ 0\ 0\ 0\ 0)$, $\mathbf{e}_1 = (1\ 0\ 0\ 0\ 0\ 0\ 0)$, $\mathbf{e}_2 = (0\ 0\ 0\ 0\ 1\ 0\ 0)$, $\mathbf{e}_3 = (0\ 1\ 1\ 0\ 0\ 0\ 0)$.

Der Dekodierer empfängt dann die Nachrichtenblöcke

$$\mathbf{r}_0 = \mathbf{v}_0 +_2 \mathbf{e}_0 = (0\ 1\ 0\ 1\ 0\ 1\ 1)$$

$$\mathbf{r}_1 = \mathbf{v}_1 +_2 \mathbf{e}_1 = (0\ 1\ 0\ 0\ 1\ 1\ 0)$$

$$\mathbf{r}_2 = \mathbf{v}_2 +_2 \mathbf{e}_2 = (1\ 0\ 0\ 1\ 0\ 0\ 1)$$

$$\mathbf{r}_3 = \mathbf{v}_3 +_2 \mathbf{e}_3 = (1\ 0\ 0\ 0\ 0\ 0\ 1).$$

Fehlerkorrektur

Die Syndromtabelle erzeugen wir systematisch, indem wir für jeden Fehlervektor $\mathbf{e}_i$ (einschließlich dem Nullvektor) mittels $\mathbf{s}_i = \mathbf{e}_i \cdot \mathbf{H}^T$ dessen Syndrom berechnen. Dies ergibt die folgende Syndromtabelle:

Fehlervektor **e**	Syndrom **s**
(0 0 0 0 0 0 0 0)	(0 0 0)
(1 0 0 0 0 0 0 0)	(1 0 0)
(0 1 0 0 0 0 0 0)	(0 1 0)
(0 0 1 0 0 0 0 0)	(0 0 1)
(0 0 0 1 0 0 0 0)	(1 1 0)
(0 0 0 0 1 0 0 0)	(1 0 1)
(0 0 0 0 0 1 0 0)	(0 1 1)
(0 0 0 0 0 0 1 0)	(1 1 1)

Da die gewählten Fehlervektoren Einheitsvektoren sind, ist die rechte Seite der Syndromtabelle gleich

$$\mathbf{I} \cdot \mathbf{H}^T = \mathbf{H}^T, \text{ plus Nullvektor.}$$

Die Syndromtabelle muss also zur 1-bit-Fehlerkorrektur gar nicht gesondert berechnet werden.

Fehlerkorrektur

Der Dekodierer liest zuerst $\mathbf{r_0} = (0\ 1\ 0\ 1\ 0\ 1\ 1)$.

Er berechnet $\mathbf{s_0} = \mathbf{r_0} \cdot \mathbf{H}^T = (0\ 0\ 0) \Leftrightarrow \mathbf{e'_0} = (0\ 0\ 0\ 0\ 0\ 0\ 0)$.

Somit wird „korrigiert“ zu $\mathbf{u_0}^* = \mathbf{r_0} +_2 \mathbf{e'_0} = (0\ 1\ 0\ 1\ 0\ 1\ 1)$.
Die Nutzbits sind die letzten $k = 4$ bits, also projiziert der Dekodierer auf $\mathbf{u_0}' = (1\ 0\ 1\ 1) = \mathbf{u_0}$.

Für $\mathbf{r_1} = (0\ 1\ 0\ 0\ 1\ 1\ 0)$ wird $\mathbf{s_1} = \mathbf{r_1} \cdot \mathbf{H}^T = (1\ 0\ 0)$ berechnet.

$\mathbf{s_1} \Leftrightarrow \mathbf{e'_1} = (1\ 0\ 0\ 0\ 0\ 0\ 0) \Rightarrow \mathbf{u_1}^* = \mathbf{r_1} +_2 \mathbf{e'_1} = (1\ 1\ 0\ 0\ 1\ 1\ 0)$.

Damit wird $\mathbf{u_1}' = (0\ 1\ 1\ 0) = \mathbf{u_1}$ erkannt.

Fehlerkorrektur

Für $\mathbf{r}_2 = (1\ 0\ 0\ 1\ 0\ 0\ 1)$ wird $\mathbf{s}_2 = \mathbf{r}_2 \cdot \mathbf{H}^T = (1\ 0\ 1)$ berechnet.

$\mathbf{s}_2 \Leftrightarrow \mathbf{e}'_2 = (0\ 0\ 0\ 0\ 1\ 0\ 0) \Rightarrow \mathbf{u}_2^* = \mathbf{r}_2 +_2 \mathbf{e}'_2 = (1\ 0\ 0\ 1\ 1\ 0\ 1)$.

Damit wird $\mathbf{u}_2' = (1\ 1\ 0\ 1) = \mathbf{u}_2$ erkannt.

Für $\mathbf{r}_3 = (1\ 0\ 0\ 0\ 0\ 0\ 1)$ wird $\mathbf{s}_3 = \mathbf{r}_3 \cdot \mathbf{H}^T = (0\ 1\ 1)$ berechnet.

$\mathbf{s}_3 \Leftrightarrow \mathbf{e}'_3 = (0\ 0\ 0\ 0\ 0\ 1\ 0) \Rightarrow \mathbf{u}_3^* = \mathbf{r}_3 +_2 \mathbf{e}_3 = (1\ 0\ 0\ 0\ 0\ 1\ 1)$.

Damit ist $\mathbf{u}_3' = (0\ 0\ 1\ 1) \neq \mathbf{u}_3$. Dekodierungsfehler aufgrund von zwei Bitfehlern!

Fehlerkorrektur

Zusammenfassung des Kodierungsbeispiels:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

u	**v = u · G**	**e**	**r = v +$_2$ e**	**s = r · H^T**	**u* = r +$_2$ e′**
1 0 1 1	0 1 0 1 0 1 1	0 0 0 0 0 0 0	0 1 0 1 0 1 1	0 0 0	0 1 0 1 0 1 1
0 1 1 0	1 1 0 0 1 1 0	1 0 0 0 0 0 0	0 1 0 0 1 1 0	1 0 0	1 1 0 0 1 1 0
1 1 0 1	1 0 0 1 1 0 1	0 0 0 0 1 0 0	1 0 0 1 0 0 1	1 0 1	1 0 0 1 1 0 1
0 0 0 1	1 1 1 0 0 0 1	0 1 1 0 0 0 0	1 0 0 0 0 0 1	0 1 1	1 0 0 0 0 1 1
Kodierer		Kanal		Dekodierer	

Fehlerkorrektur

► *Sei* **u** = (1 0 1 0), **e** = (0 0 0 1 0 0 0)
und **G** *und* **H** *wie zuvor.*

Berechnen Sie **v**, **r**, **s** *und* **u***.

Fehlerkorrektur

Wir sind bislang von **gleichverteilten**, **unabhängigen** Fehlern ausgegangen.

Ein Fehler wäre dann zu jedem Zeitpunkt und Ort gleichwahrscheinlich und würde nicht davon abhängen, ob in seiner Nachbarschaft Fehler aufgetreten sind.

Das ist eine in der Statistik übliche, aber in der Informationsübertragung häufig nicht zutreffende Annahme.

Sind diese Voraussetzungen nicht erfüllt, d.h. treten Fehler gebündelt (in **Fehlerbüscheln**, ***Bursts***) auf, so sollte man andere Codes oder *Interleaving* (Verzahnung, Verschränkung) verwenden.

Fehlerkorrektur

Die gängigste Methode zur Verzahnung ist das ***Block Interleaving***.

Dabei werden q verschiedene Codeworte reihenweise zu einer Matrix zusammengefasst und dann spaltenweise gesendet, wie folgt:

Codewort 1	0	1	1	**1**	0	1	0
Codewort 2	1	1	0	**1**	1	0	0
...							
...							
...							
Codewort q	1	1	1	0	0	0	1

Zuerst wird die grün unterlegte linke Spalte gesendet, dann die zweite Spalte, usw.

Die pink unterlegten Felder in der vierten Spalte stellen einen zeitlich / räumlich zusammenhängenden Fehlerbüschel dar.

Einzelne Fehlerbüschel mit einer Länge $\leq q$ verursachen dann lediglich einen Fehler je Codewort und können somit korrigiert werden.

Fehlerkorrektur

Eine hohe Kodierungsgeschwindigkeit ist wichtig.
Folgende Optionen bieten sich an:

- Kodierung mittels Matrix-Multiplikationen in Software.

Die langsamste Methode, in der Praxis ungeeignet.

- Kodierung mittels spezieller Hardware. Schnell, teuer.
- Kodierung mittels vorberechneter Tabellen (*look-up tables*).

Man berechnet für sämtliche Blöcke die zugehörigen Kodierungen bzw. Syndrome vorab und speichert sie in einer Tabelle.

Die Kodierung besteht dann nur noch aus einem Zugriff auf den passenden Tabellen-Eintrag.

Das ist schnell, aber der Speicherbedarf steigt exponentiell mit k (für $\mathbf{v} = \mathbf{u} \cdot \mathbf{G}$) bzw. m (für $\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T$).

Fehlerkorrektur

Ein linearer Blockcode heißt **zyklisch**, wenn für jedes gültige Codewort **v** sämtliche zyklischen Verschiebungen ebenfalls gültige Codewörter sind.

Ist also $\mathbf{v} = (v_0\ v_1\ v_2\ \ldots\ v_{k-1})$ ein Codewort, so sind auch

$\mathbf{v}' = (v_1\ v_2\ \ldots\ v_{k-1}\ v_0)$ und

$\mathbf{v}'' = (v_2\ \ldots\ v_{k-1}\ v_0\ v_1)$ etc. gültige Codeworte.

Zyklische Codes sind gegenüber den nicht-zyklischen Blockcodes

- leistungsfähiger (d.h. niedrigere Fehlerwahrscheinlichkeit bei gleicher Coderate bzw. höhere Coderate bei gleicher Fehlerwahrscheinlichkeit)
- leichter/effizienter zu kodieren und dekodieren.

Fehlerkorrektur

Zyklische Codes erlauben die Darstellung von Codeworten als Koeffizienten von binären Polynomen, z.B.

$$(1\ 1\ 0\ 1\ 0\ 0\ 0) \leftrightarrow 1 + x + x^3$$

Man legt ein **Generatorpolynom** $\mathbf{g}(x)$ fest.

Alle gültigen Codeworte müssen ein Vielfaches von $\mathbf{g}(x)$ sein.

Für $\mathbf{g}(x) = 1 + x + x^3$ sind gültige Codeworte der Länge 7 z.B.

$0 \cdot \mathbf{g}(x) = (\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ )$

$1 \cdot \mathbf{g}(x) = (\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ )$

$x \cdot \mathbf{g}(x) = x + x^2 + x^4 = (\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ )$

$(1 + x^3) \cdot \mathbf{g}(x) = 1 + x + x^4 + x^6 = (\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ )$

Fehlerkorrektur

Zur Kodierung einer Nachricht erweitert man die Datenbits so, dass das dadurch entstehende Codewort ein Vielfaches von **g**(x) ist.

Die Dekodierung erfolgt umgekehrt über Polynomdivision:

Man teilt die empfangene Nachricht **r**(x) durch **g**(x).
Ist der Rest (= Syndrom) gleich null, so ist die Nachricht fehlerfrei, ansonsten erfolgt die Fehlerkorrektur über die Syndromtabelle.

BCH Codes (Bose / Chaudhuri / Hocquenghem, 1959) sind zyklische Codes, die auf einem Minimalpolynom (d.h. auf einem Polynom mit möglichst niedrigem Grad) aufbauen.

Mit BCH Codes lassen sich mehrere Fehler korrigieren.

Der Hamming Code ergibt sich als Spezialfall, bei dem nur ein Fehler korrigiert wird.

Fehlerkorrektur

Für BCH Codes wählt man immer eine Codewortlänge der Form $n = 2^i - 1,\ i \geq 3$.

Will man t Fehler korrigieren, so muss wie oben gesehen für den Abstand zwischen den Codeworten $d_{min} \geq 2t + 1$ gelten. Dies wird von den BCH Codeworten erreicht.

Für die Anzahl der Prüfbits $m = n - k$ gilt die Abschätzung $m \leq i \cdot t$.

Die Tabelle zeigt die Kennzahlen für die kleinsten BCH-Codes.

n	k	t
7	4	1
15	11	1
15	7	2
15	5	3
31	26	1
31	21	2
31	16	3
31	11	4
31	6	5

Fehlerkorrektur

Reed-Solomon (RS) Codes sind nicht-binäre BCH Codes.
Sie gehören zu den leistungsfähigsten bekannten Codes.

Zur Dekodierung von RS Codes können verschiedene Algorithmen verwendet werden, die unterschiedliche Strategien zur Fehlerkorrektur verfolgen und somit auch unterschiedliche Ergebnisse liefern.

Eine Variante der RS Codes, genannt **CIRC** (*Cross Interleaved Reed-Solomon Code*), kommt auf Audio-CDs zum Einsatz.

Interleaving (**Verzahnung, Verschränkung**): Aufeinander folgende Nachrichtenbits werden vor dem Senden/Speichern umsortiert. $\Rightarrow$ Zusammenhängende Fehlerbüschel werden auf verschiedene Blöcke verteilt und können einzeln korrigiert werden.

Der Empfänger führt entsprechend das De-Interleaving durch.

Fehlerkorrektur

Fehlerkorrigierende Codes kommen manchmal auch im Hauptspeicher zum Einsatz:

Für ein 64-Bit breites ECC-DIMM (*Dual Inline Memory Module*) verwendet man 8 zusätzliche Bits für die Fehlerkorrektur, so dass man insgesamt 72 Datenleitungen hat. Die Berechnung der 8 Prüfbits erfolgt direkt durch den Speichercontroller.

ECC-DIMMs führen zu erhöhtem Zeitaufwand beim Booten (Speichertest) und zur Laufzeit (Überwachung/Protokollierung von Fehlern).

Die ECC-Speichertechnik lässt sich RAID-ähnlich aufbauen (RAID = *Redundant Array of Independent Disks*), z.B. durch Verteilung der Prüfbits über vier DIMMs.