

2 Tutorium Digitaltechnik

2.1 Optimale Codes:

Optimale Codes versuchen die im Mittel auftretende durchschnittliche Codewortlänge \bar{m} zu minimieren. Sie bestehen also aus Codewörtern unterschiedlicher Länge.

- Mittlere Codewortlänge:

$$\bar{m} = \sum_{i=1}^n p(x_i) * m(x_i)$$

$p(x_i)$: Auftrittswahrscheinlichkeit des Codewortes

$m(x_i)$: Anzahl der Bits des Codewortes

- Entropie:
Ermöglicht Bewertung eines Codes. Desto näher \bar{m} bei der Entropie der Quelle H liegt, desto besser ist der Code.

$$H = \sum (p(x_i) * \lg(\frac{1}{p(x_i)}))$$

- Vergleich Huffman \leftrightarrow Shanon-Fano: Was ist besser?
Es gilt immer: $\bar{m}_{Huffman} \leq \bar{m}_{Shanon-Fano}$
Außerdem ist die Codierung nach Huffman stets eindeutig.
- Besonderheit der optimalen Codes (Huffman/Shanon-Fano): Sie sind präfixfrei, d.h. keines der Codewörter beginnt mit einem anderen Codewort. Dadurch sind die einzelnen Codewörter trotz unterschiedlicher Länge abgrenzbar.

2.2 Shanon-Fano-Codierung:

1. Sortieren Sie die Elemente entsprechend ihrer Auftrittswahrscheinlichkeiten aufsteigend von links nach rechts.
Falls sinnvoll Abkürzungen (Buchstaben) für Zustände einführen.
2. Bilden Sie 2 Teilmengen aus der Menge, sodass die Summen der Auftrittswahrscheinlichkeiten der 2 Teilmengen möglichst gleich werden.
(Reihenfolge dabei gleich lassen)
3. Wiederholen Sie Schritt 2 für die entstehenden Teilmengen solange, bis die Teilmengen nur noch ein Element enthalten.
4. Weisen Sie entsprechend der Aufgabe jeweils den linken und rechten Ästen des entstehenden Baumes die "0" bzw. "1" zu.

2.3 Huffman-Codierung:

1. Sortieren Sie die Elemente entsprechend ihrer Auftrittswahrscheinlichkeiten aufsteigend von links nach rechts.
(bei gleichen AWS alphabetisch)
2. Verschmelzen Sie die beiden Elemente mit der kleinsten AWS zu einem neuen Element.
3. Sortieren Sie dieses neue Element entsprechend seiner Wahrscheinlichkeit in die Menge ein.
4. Wiederholen Sie Schritt 2 und 3, bis Sie nur noch ein Element in ihrer Menge haben.
5. Weisen Sie entsprechend der Aufgabe jeweils den linken und rechten Ästen des entstehenden Baumes die "0" bzw. "1" zu.

2.4 Blocksicherung und Scrambling:

- Bündelfehler:
Bei einer länger andauernden Signalstörung passiert es häufig, dass mehrere Bits nacheinander verfälscht werden. Dies ist durch Paritätssicherung schlecht zu erkennen.
- Scrambling:
Um Bündelfehler zu erkennen werden beim Scrambling Codeworte nicht nacheinander übertragen, sondern Spaltenweise. Dadurch ist es mit einem Paritätsbit am Ende jedes CW's möglich Bündelfehler bis zur Länge der Zeilenzahl eines Blocks zu identifizieren.

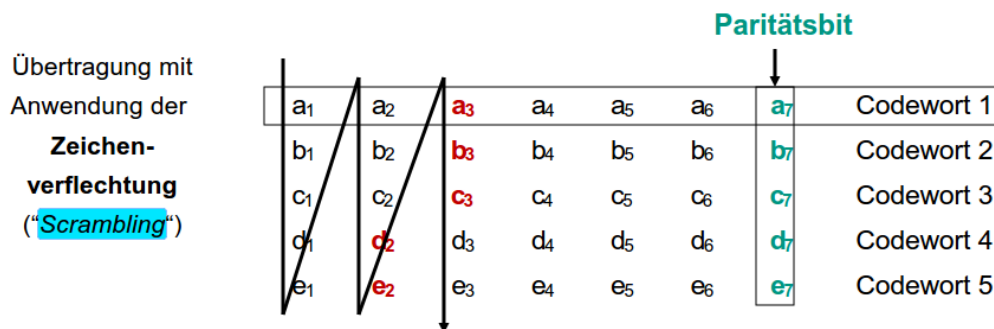


Figure 1: Beispiel zu Scrambling bei einem fünfzeiligen Block

- Overhead:
Der Anteil des Codes, der keine Informationen enthält, also nur zur Absicherung vor Fehlern mitgesendet wird. Anteil an Overhead:

$$\text{Overhead} = \frac{\text{Sicherungsbits}}{\text{Nutzbits}}$$