5.3. Entropy Coding

- The different probabilities for the appearing of single symbols are used
  - to shorten the average code length by assigning shorter codes to more probable symbols => Morse-, Huffman-, Arithmetic Code
  - to simplify the encoding/decoding by assigning simpler codes to more probable symbols => e.g. the braille
- Entropy coding schemes are lossless compression schemes.
- Entropy coding procedures rely on statistically independent information events to produce optimal results (maximum theoretical compression).

Remark:
A **prefix code** (or prefix-free code) is a code in which no codeword is a prefix to another codeword. This enables unique decodability with variable code length without any separator symbol.

The Braille

Grund-form

A B C D E

F G H I J

K L M N O

P Q R S T

Ü V X Y Z

Artikel bei Wikipedia

Morse Code

Example: Letter statistic compared to Morse Code

Letter	probability German	probability English	Morse Code
e	16.65%	12.41%	.
n	10.36%	6.41%	_ .
i	8.14%	6.46%	. .
t	5.53%	8.90%	_
a	5.15%	8.09%	. _
o	2.25%	8.13%	_ _ _
x	0.03%	0.20%	_ . . _
y	0.03%	2.14%	_ . _ _

- variable code length & nonprefix code

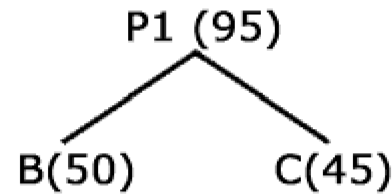
→ needs separator symbol for unique decodability
  - Example: "..-…" → eeteee, ini, ...

Huffman Coding (1)

- Idea: Use short bit patterns for frequently used symbols
  - Sort all symbols by probability
  - Get the two symbols with the lowest probability, remove them from the list and insert a "parent symbol" to the sorted list where the probability is the sum of both symbols
  - If there are at least two elements left, continue with step 2
  - Assign 0 and 1 to the branches of the tree
- **Example:** 1380 bits are used for "regular coding"

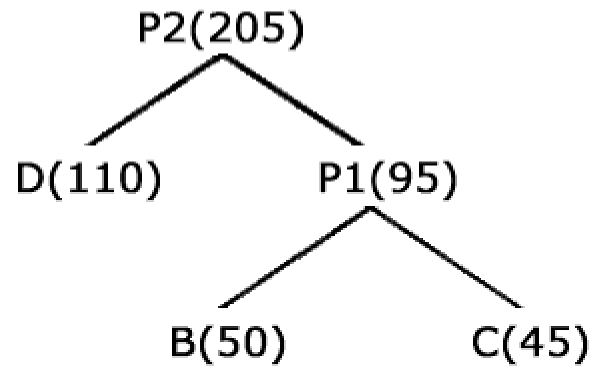
Symbol	Count	regular
E	135	000
A	120	001
D	110	010
B	50	011
C	45	100

Huffman Coding (2)



Symbol	Count
E	135
A	120
D	110
P1	95

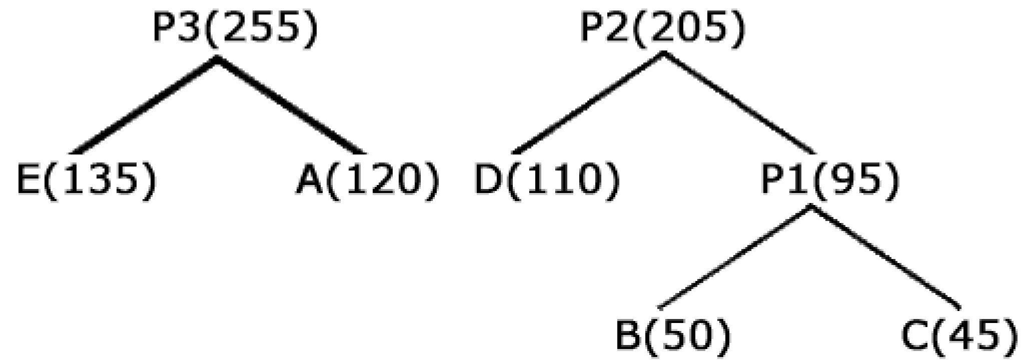
Huffman Coding (3)



Symbol	Count
P2	205
E	135
A	120

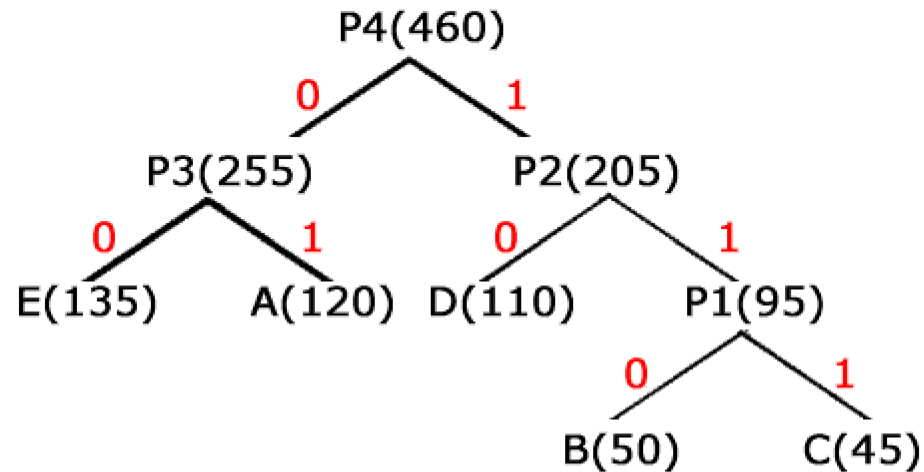
http://www.icsy.de

Huffman Coding (4)



Symbol	Count
P3	255
P2	205

Huffman Coding (5)



Symbol	Count	Huffman	p_i
E	135	00	0,294
A	120	01	0,261
D	110	10	0,239
B	50	110	0,109
C	45	111	0,098

→ 1015 bits are used for huffman coding

Integrated Communication Systems

http://www.icsy.de

Huffman Coding (6)

Symbol	Count	Huffman	p_i
E	135	00	0,2935
A	120	01	0,2609
D	110	10	0,2391
B	50	110	0,1087
C	45	111	0,0978

- Entropy:

$$H = \eta = \sum_i p_i \cdot \log_2 \frac{1}{p_i} \quad \text{[bit]}$$

=> $H \approx 2{,}1944$

- Average code length:
$l = 2^*p_1+2^*p_2+2^*p_3+3^*p_4+3^*p_5 \approx 2{,}2065$

Adaptive Huffman Coding (1)

- **Problem:**
  - The previous algorithm requires statistical knowledge (the probability distribution) which is often not available (e.g. live audio, video).
  - Even when statistical knowledge is available, it could be too much overhead to send it.
- **Solution:**
  - use of adaptive algorithms
  - here: Adaptive Huffman Coding which keeps up with estimating the required probability distribution from previously encoded/decoded symbols by an update procedure

http://www.icsy.de

Integrated Communication Systems

Adaptive Huffman Coding (2)

http://www.icsy.de

Encoder

```
Initialize_model();
while (( c = getc(input))
!= eof)
  {
    encode (c, output);
    update_model(c);
  }
```

Decoder

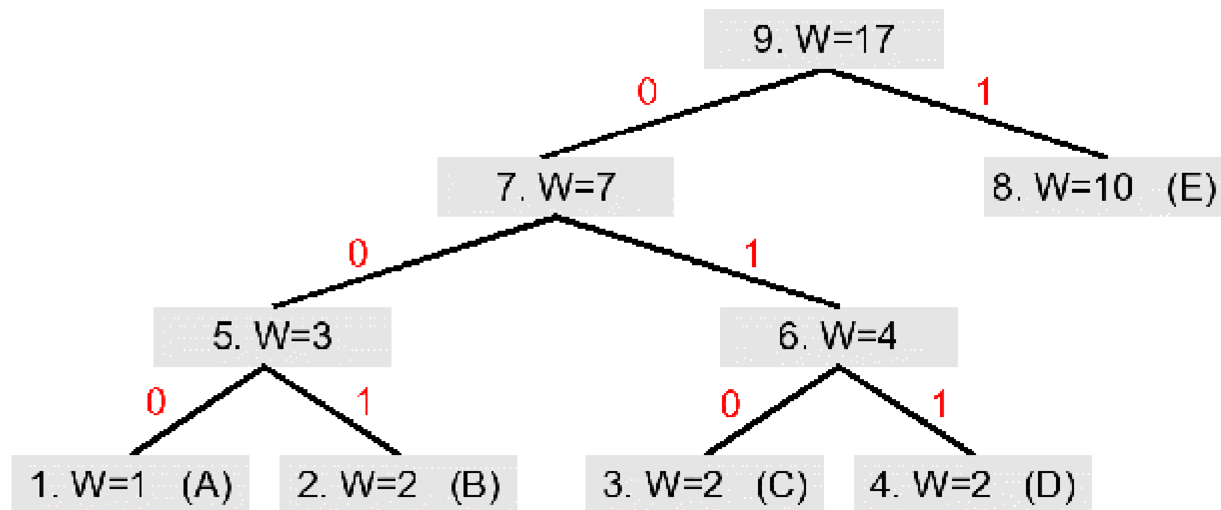
```
Initialize_model();
while (( c = decode(input))
!= eof)
  {
    putc (c, output);
    update_model(c);
  }
```

Adaptive Huffman Coding (3)

- Important: encoder and decoder have to use exactly the same *initialization* and *update_model* routines
- *update_model* does two things:
  - increment the count
  - update the resulting Huffman tree
    - during the updates, the Huffman tree will be maintained its sibling property, i.e. the nodes are arranged in order of increasing weights
    - when swapping is necessary, the farthest node with weight W is swapped with the node whose weight has just been increased to W+1 (If the node with the weight W has a subtree beneath it, then the subtree will go with it)

Adaptive Huffman Coding (4)

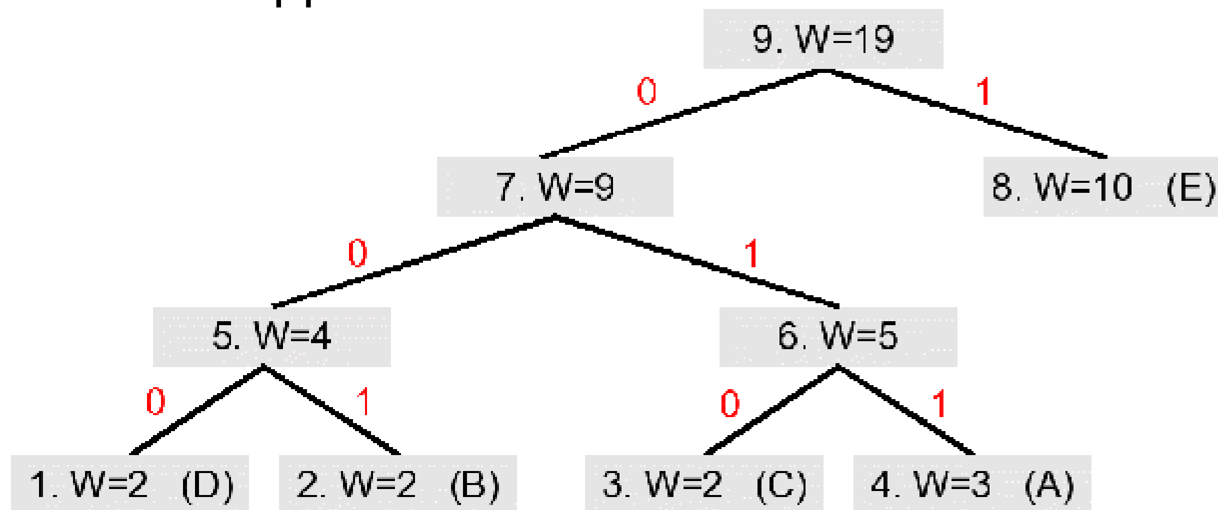
- Example n-th Huffman tree



- resulting code for A: 000

Adaptive Huffman Coding (5)

- n+2-th Huffman tree
  - A was incremented twice node
  - A and D swapped

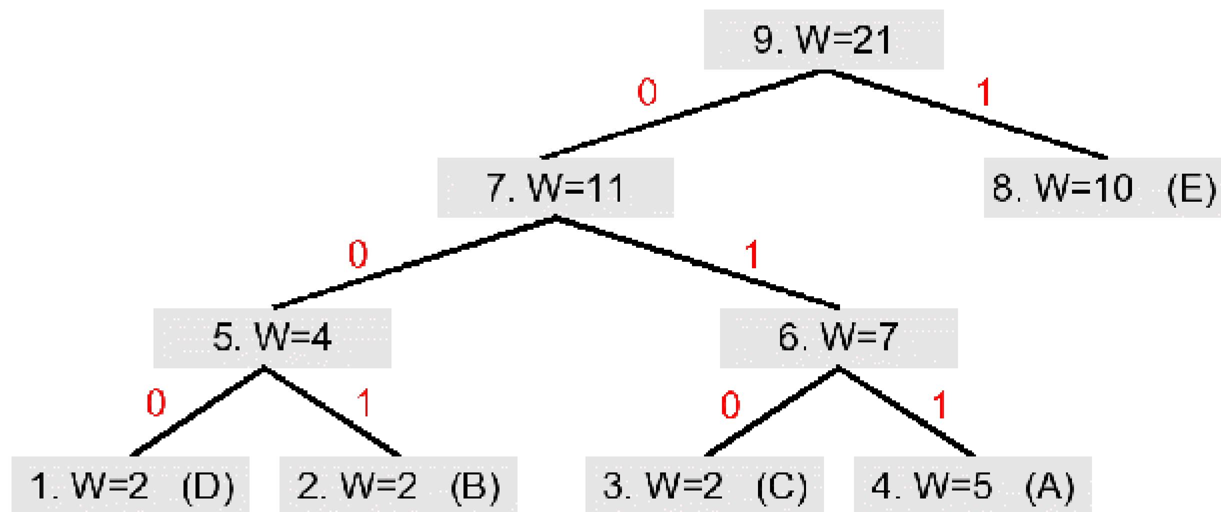


- new code for A: 011

http://www.icsy.de

Adaptive Huffman Coding (6)

- A was incremented twice



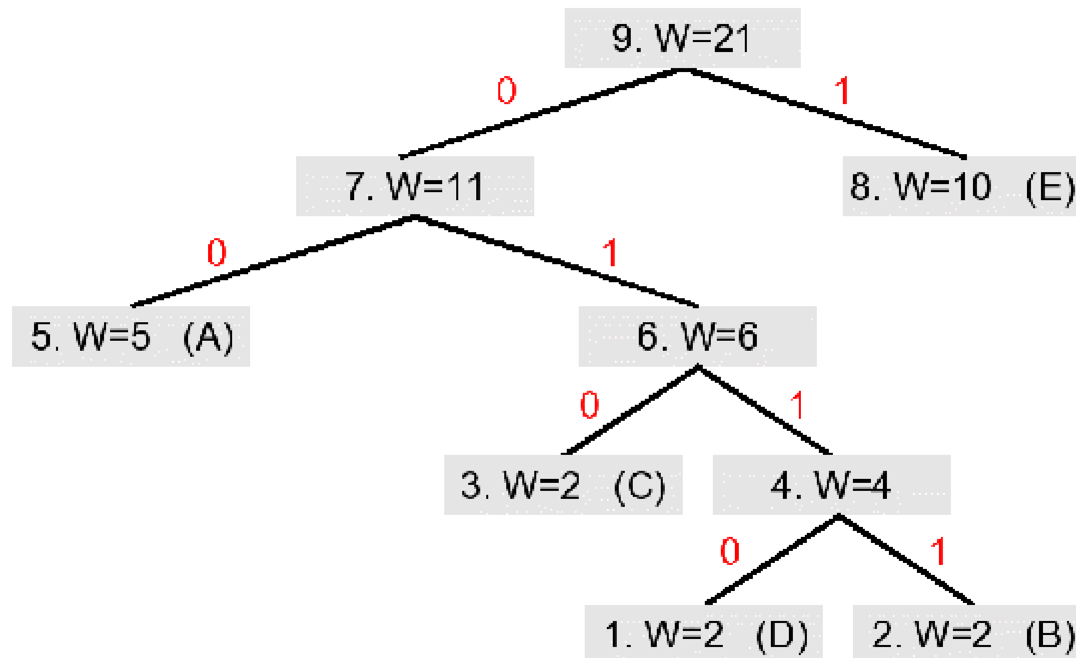
- The 4th (A) and the 5th node have to swap.

Technische Universität Kaiserslautern

ICSY Integrated Communication Systems

Adaptive Huffman Coding (7)

- resulting tree after 1st swap

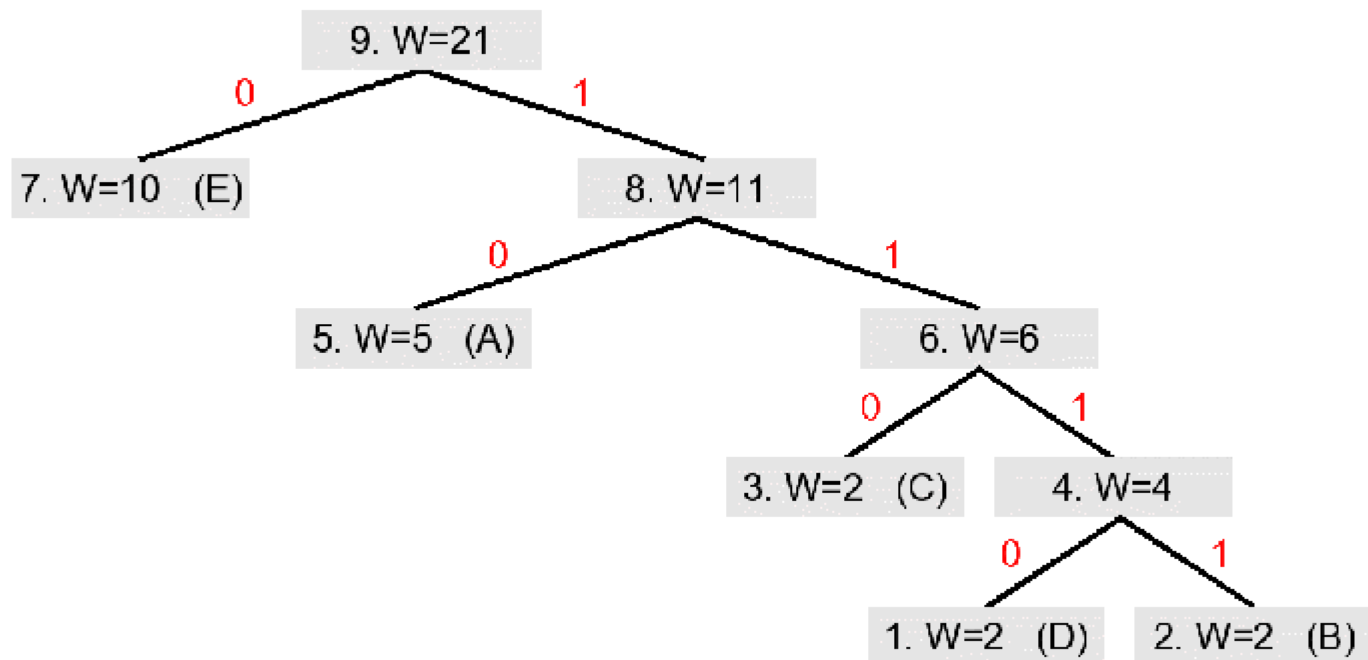


- The 8th (E) and the 7th node have to swap.

http://www.icsy.de

Adaptive Huffman Coding (8)

- n+4-th Huffman tree



- new code for A: 10

Huffman Coding Evaluation (1)

- **Advantages**
  - The algorithm is simple.
  - Huffman Coding is optimal according to information theory when the probability of each input symbol is a negative power of two.
  - If p_{max} is the largest probability in the probability model, then the upper bound is H + p_{max} + ld[(2 ld e)/e] = H + p_{max} + 0,086. (see proof of Gallager)

    This is a tighter bound than $H \leq \bar{l} < H + 1$

http://www.icsy.de

Huffman Coding Evaluation (2)

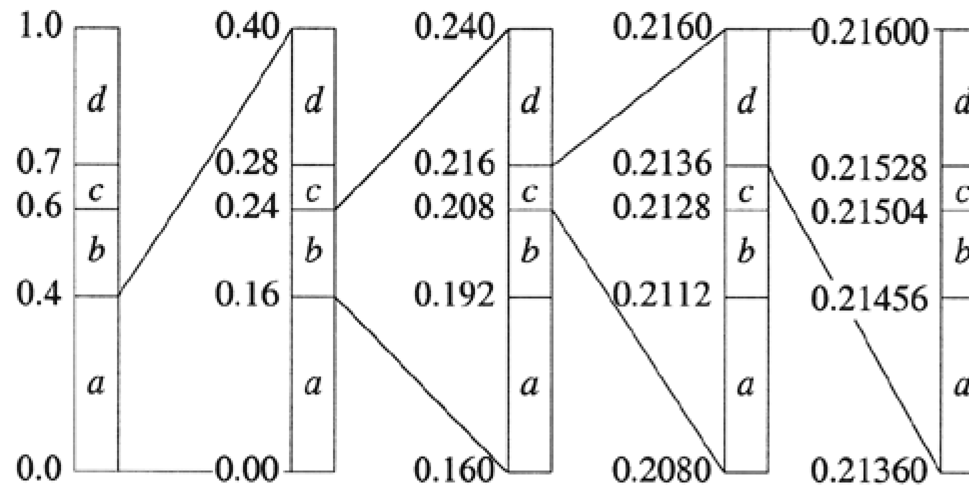
- **Limitations:**
  - Huffman is designed to code single characters only. Therefore at least one bit is required per character, e.g. a word of 8 characters requires at least an 8 bit code
  - Not suitable for strings of different length or changing probabilities for characters in a different context, respectively
    Examples for both interpretations of that problem:
    - Huffman coding does not support different probabilities for: "c" "h" "s" and "sch"
    - For a usual german text p("e") > p("h") is valid, but if the preceding characters have been "sc" then p("h") > p("e") is valid
  - (non-satisfying) solutions for both scenarios:
    - Solution: Use a special coding where "sch" is one character only, but this requires knowledge about frequent character combinations in advance
    - Solution: use different huffman codes with respect to the context, but this leads to large code tables which must be appended to the coded data

Arithmetic Coding (1)

- **Arithmetic coding generates codes that are optimal according to information theory, supports average code word length smaller 1 and strings of different length.**
- **The probability model is separated from the encoding operation:**
  - Probability model: Encode strings and not single characters. Assign a "codeword" to each possible string. A codeword consists of a half open subinterval of the interval [0,1). Subintervals for different strings do not intersect.
  - Encoding: a given subinterval can be uniquely identified by any value of that interval. Use the value with the shortest nonambiguous binary representation to identify the subinterval.
- **In practice, the subinterval is refined incrementally using the probabilities of the individual events.**
  - If strings of different length are used then extra information is needed to detect the end of a string

Arithmetic Coding (2)

- **Example I:**
  - Four characters *a, b, c, d* with probabilities *p*(*a*)=0.4, *p*(*b*)=0.2, *p*(*c*)=0.1, *p*(*d*)=0.3. Codeword assignment for string "abcd":



→ codeword (subinterval): [0.21360, 0.21600)

http://www.icsy.de

Technische Universität Kaiserslautern

ICSY Integrated Communication Systems

Arithmetic Coding (3)

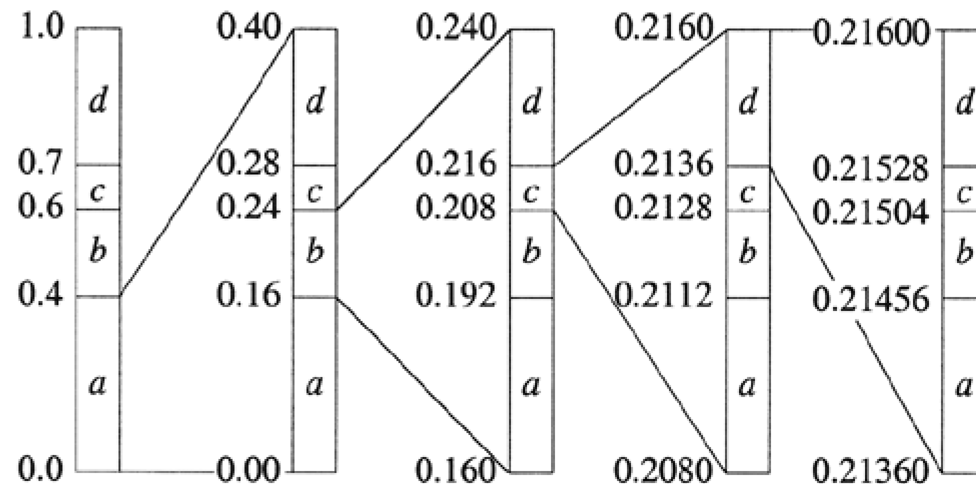
- **Example I: (cont.)**
  - Encoding:
    Any value within the subinterval [0.21360, 0.21600) will now be a well-defined identifier of the string “abcd".

		Bit	Sum
2^{-1}	0.5	0	0
2^{-2}	0.25	0	0
2^{-3}	0.125	1	0.125
2^{-4}	0.0625	1	0.1875
2^{-5}	0.03125	0	
2^{-6}	0.015625	1	0.203125
2^{-7}	0.0078125	1	0.2109375
2^{-8}	0.00390625	1	0.21484375

  - "0.00110111" will be the shortest binary representation of the subinterval.

Arithmetic Coding (4)

- **Example I: (cont.)**
  - Decoding: "0.00110111" resp. 0.21484375



  - Receiver can rebuild the subinterval development but:
    code is member of any further subinterval of [0.21360, 0.21600)
    => extra symbol necessary to mark end of string

Arithmetic Coding (5)

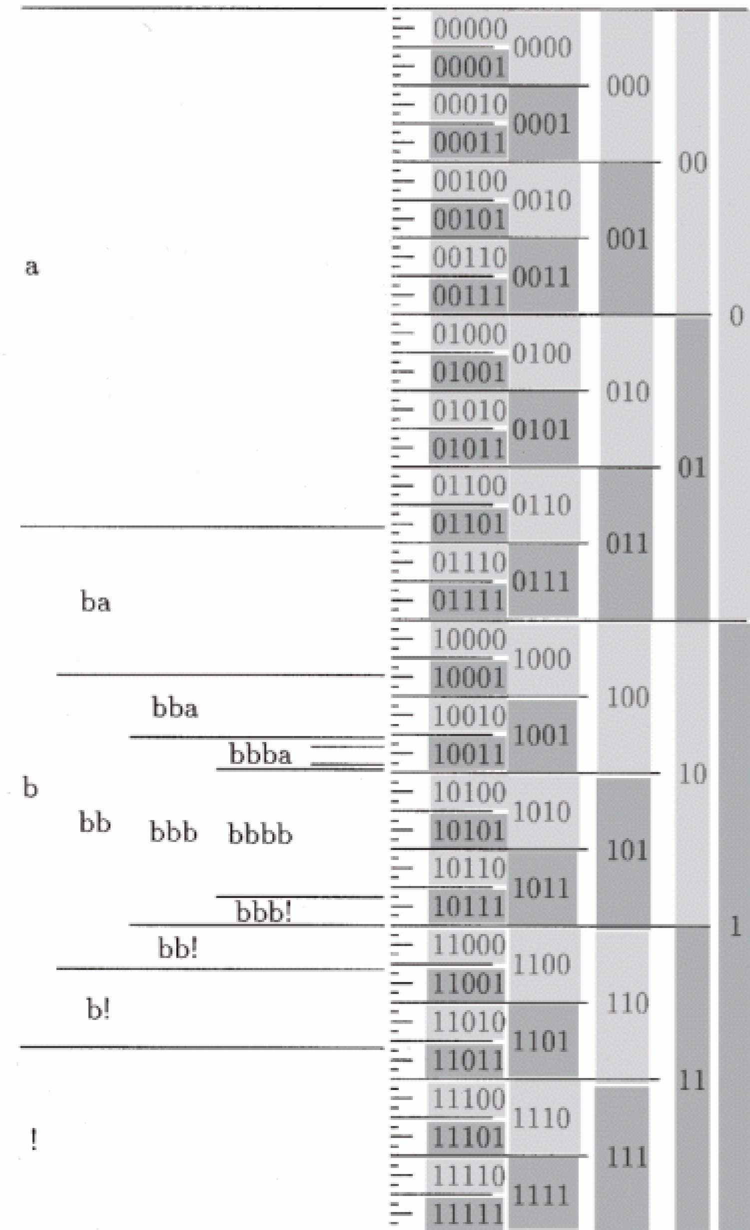
- Example II
  - Propabilties:

probailities

$P(a)=0.425$
$P(a|b)=0.283$
$P(a|bb)=0.212$
$P(a|bbb)=0.170$
$P(a|bbba)=0.283$

$P(b)=0.425$
$P(b|b)=0.567$
$P(b|bb)=0.637$
$P(b|bbb)=0.680$
$P(b|bbba)=0.567$

$P(!)=0.150$
$P(!|b)=0.150$
$P(!|bb)=0.150$
$P(!|bbb)=0.150$
$P(!|bbba)=0.150$

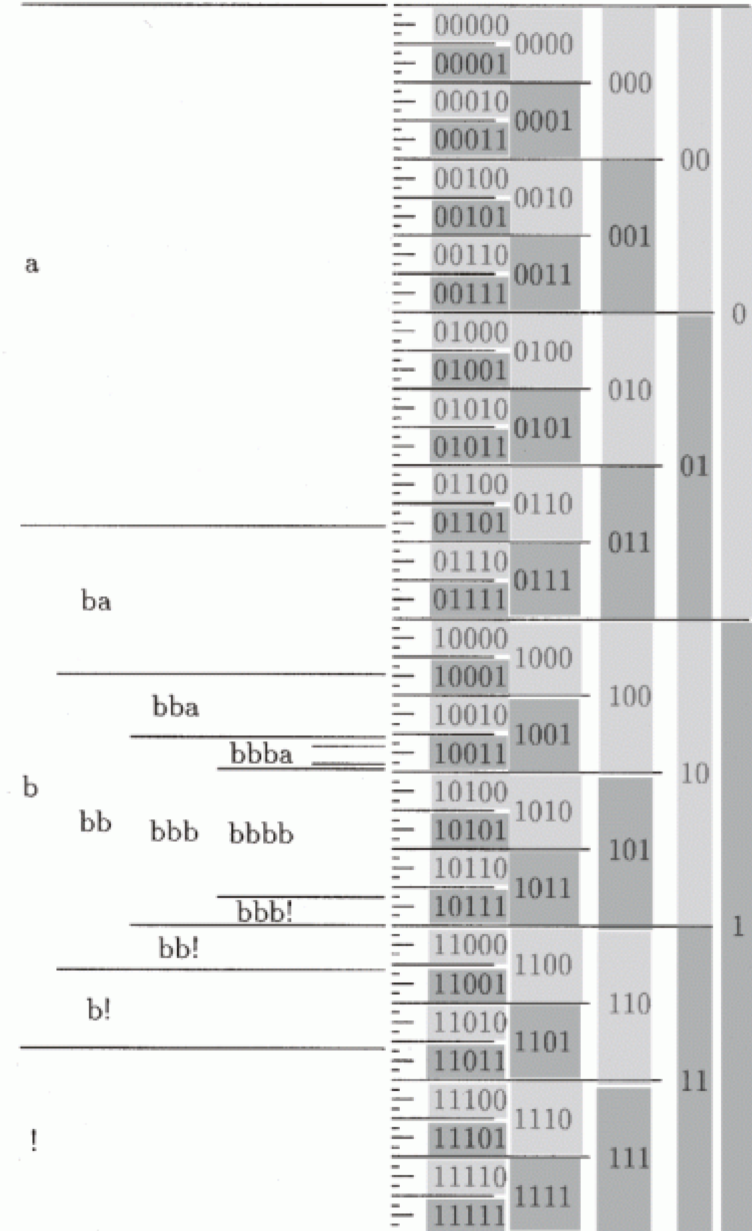


Arithmetic Coding (6)

Example II: (cont.)

- Termination rules:
  - "a" and "!" are termination symbols
  - Strings starting with "b" have a length of max 4.
- Decode: 10010001010

Bit	Output
1	-
0	b
0	-
1	b
0	a
0	-
0	a
1	-
0	b
1	bb
0	b



http://www.icsy.de

Arithmetic Coding (7)

Remarks:

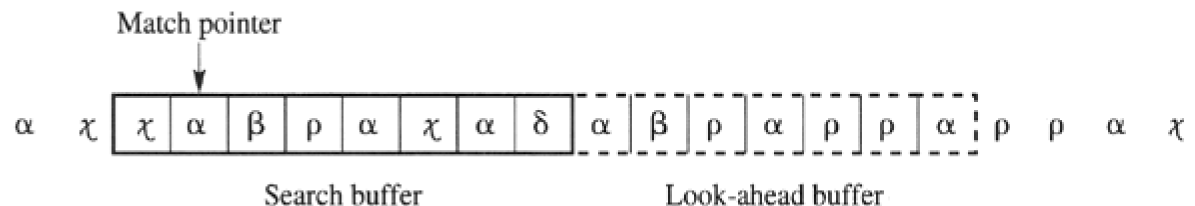
- The more characters a string has, the better the result concerning average code word length.
- Implementing arithmetic coding is more complex than huffman coding.
- Alphabets with *k* characters and strings with length *m* destroy the idea of huffman coding for increasing *m* (codebook with size *k^m*). Arithmetic coding can be adapted.
- If the alphabet is small and the probabilities are highly unbalanced arithmetic coding is superior to huffman coding.
- Efficient arithmetic coding implementations exclusively rely on integer arithmetic.
- United States Patent 4,122,440; Method and means for **arithmetic string coding**; International Business Machines Corporation; October 24, 1978
- A precise comparison between arithmetic coding and huffman coding can be found in [Sayood].

http://www.icsy.de

Lempel - Ziv (LZ77) (1)

- **Algorithm for compression of character sequences:**
  - **assumption:** sequences of characters are repeated
  - **idea:** replace a character sequence by a reference to an earlier occurrence

1. **Define a**
   - search buffer = (portion) of recently encoded data
   - look-ahead buffer = not yet encoded data
2. **Find the longest match between**
   - the first characters of the look ahead buffer
   - and an arbitrary character sequence in the search buffer
3. **Produces output <offset, length, next_character>**
   - offset + length = reference to earlier occurence
   - next_character = the first character following the match in the look ahead buffer



Lempel - Ziv (LZ77) (2)

- Example:

Pos	1	2	3	4	5	6	7	8	9	10
Char	A	A	B	C	B	B	A	B	C	A

Step	Pos	Match	Char	Output
1	1	-	A	<0,0,A>
2	2	A	B	<1,1,B>
3	4	-	C	<0,0,C>
4	5	B	B	<2,1,B>
5	7	ABC	A	<5,3,A>

Lempel - Ziv (LZ77) (3)

- **Remarks:**
  - the search and look ahead buffer have a limited size
    - the number of bits needed to encode pointers and length information depends on the buffer sizes
    - worst case: the character sequences are longer than one of the buffers
    - typical size is 4 - 64 KB
  - sometimes other representations of the triple are used
    - next_char only if necessary (i.e. no match found)
    - enabling dynamic change of buffer sizes
  - LZ77 or variants are often used before entropy coding
    - LZ77 + Huffmann coding are used by "gzip" and for "PNG"
    - "PKZip", "Zip", "LHarc" and "ARJ" use LZ77-based algorithms
  - LZW Patent Information
    - Unisys U.S. LZW Patent No. 4,558,302 expired on June 20, 2003, the counterpart patents in the UK, France, Germany, and Italy expired on June 18, 2004, the Japanese counterpart patents expired on June 20, 2004 and the counterpart Canadian patent expired on July 7, 2004.
    - Unisys Corporation also holds patents on a number of improvements on the inventions claimed in the expired patents.

Lempel - Ziv - Welch (LZ78, LZW) (4)

- **LZ78:**
  - drops the search buffer and keeps an explicit dictionary (build at encoder and decoder in identical manner)
  - produces output <index, next_character>
  - example, adaptive encoding "abababa" will result in:

Step	Output	Directory
1	<0,a>	"a"
2	<0,b>	"b"
3	<1,b>	"ab"
4	<3,a>	"aba"

- **LZW:**
  - produces only output <index>
  - dictionary has to be initialized with all letters of source alphabet
  - new patterns are added to the dictionary
  - used by unix "compress", "GIF", "V24.bis", "TIFF"
- In practice, limit size of the directory.

Lempel - Ziv - Welch (LZW) (5)

http://www.icsy.de

- Example: wabbapwabbapwabbapwabbapwoopwoopwoo

Initial LZW dictionary.

Index	Entry
1	ƀ
2	*a*
3	*b*
4	*o*
5	*w*

Constructing the 12th entry of the LZW dictionary.

Index	Entry
1	ƀ
2	*a*
3	*b*
4	*o*
5	*w*
6	*wa*
7	*ab*
8	*bb*
9	*ba*
10	*a*ƀ
11	ƀ*w*
12	*w* . . .

→ Encoder output sequence: 5 2 3 3 2 1

Lempel - Ziv - Welch (LZW) (6)

Initial LZW dictionary.

Index	Entry
1	ƀ
2	*a*
3	*b*
4	*o*
5	*w*

The LZW dictionary for encoding wabbaƀwabbaƀwabbaƀwabbaƀwooƀwooƀwoo.

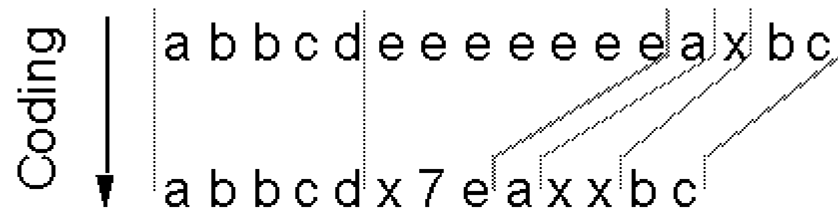
Index	Entry	Index	Entry
1	ƀ	14	*a*ƀ*w*
2	*a*	15	*wabb*
3	*b*	16	*ba*ƀ
4	*o*	17	ƀ*wa*
5	*w*	18	*abb*
6	*wa*	19	*ba*ƀ*w*
7	*ab*	20	*wo*
8	*bb*	21	*oo*
9	*ba*	22	*o*ƀ
10	*a*ƀ	23	ƀ*wo*
11	ƀ*w*	24	*oo*ƀ
12	*wab*	25	ƀ*woo*
13	*bba*		

→ Encoder output sequence: 5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4

http://www.icsy.de

Run-length Coding

- Run-length coding compresses: same successive symbols
- Example:



  - "x" marks the beginning of a coded sequence
  - special coding is required to code "x" as a symbol, here "xx" is used.
  - How to code a bit stream?
    1001011111000000010100100000000000011110
  - Zero suppression is a special form of run-length coding

5.4. Quantization (1)

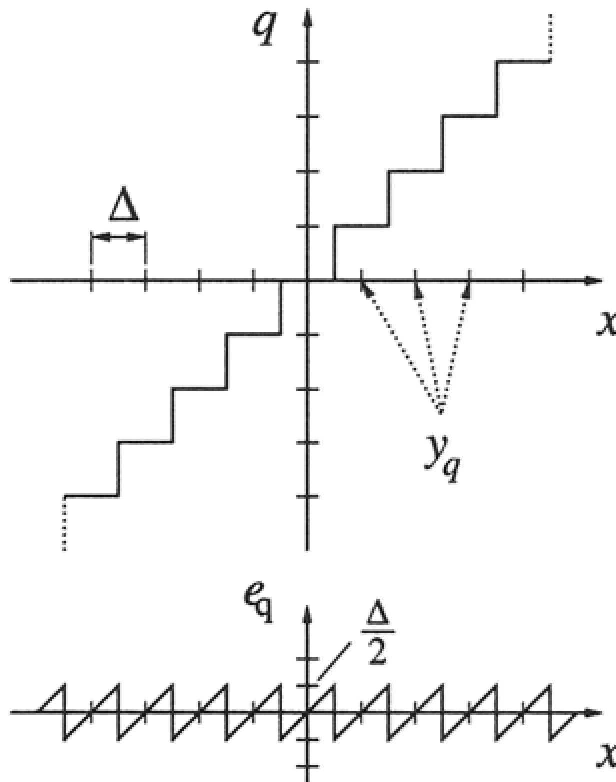
- one of the simplest and most general ideas in lossy compression
- *quantization*
  Process of representing a large - possibly infinite - set of values with a much smaller set, i.e. represent each source output using one of a small number of codewords.
- *quantizer*
  An encoder-decoder pair with
  - encoder: divides the range of values that a source generates into a number of intervals each represented by a distinct codeword (interval number) and maps all the source outputs that fall into a particular interval by the representing codeword
  - decoder: generates for every codeword a reconstruction value, that in some sense best represents all the values in the interval
- uniform versus non-uniform quantization, scalar versus vector quantization

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Integrated Communication Systems

Quantization (2)

Uniform scalar quantization



- uniform: equal interval width Δ
- **quantization** (encoding):

$$x \mapsto q, \quad x \in \mathbb{R}, \quad q \in \mathbb{Z}$$

$$q = \left\lfloor \frac{|x|}{\Delta} + \frac{1}{2} \right\rfloor \cdot \operatorname{sgn}(x)$$

- **reconstruction** (decoding):

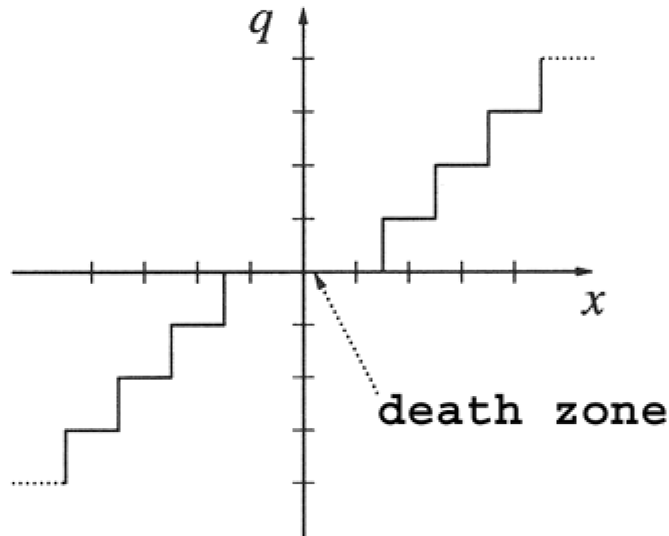
$$q \mapsto y_q, \quad \text{with} \quad [x]_Q = y_q, \quad y_q \in \mathbb{R}$$

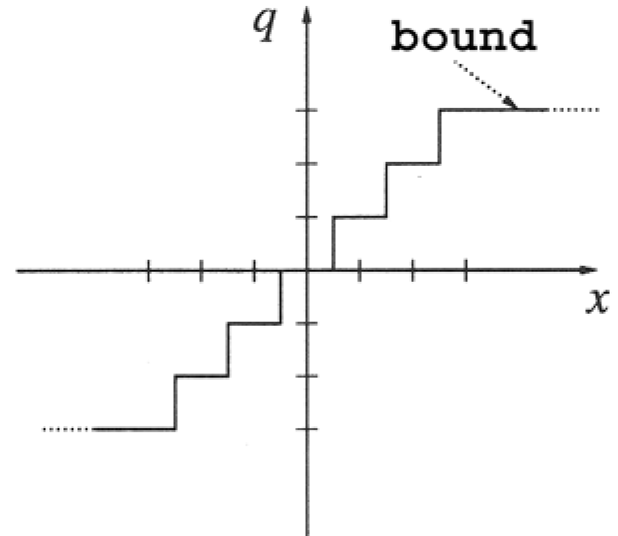
$$[x]_Q = y_q = q \cdot \Delta$$

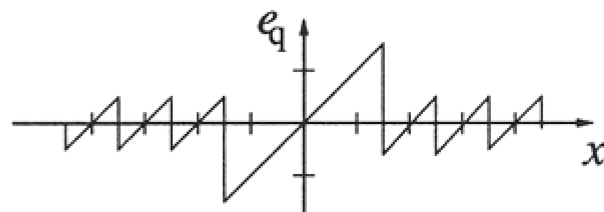
- **quantization error**: $e_q = x - [x]_Q$

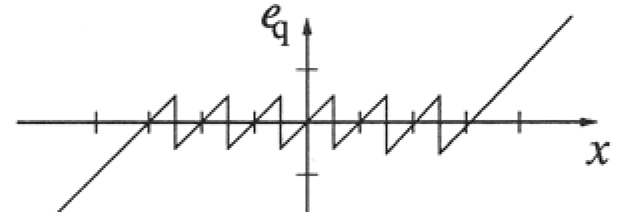
http://www.icsy.de

Quantization (3)









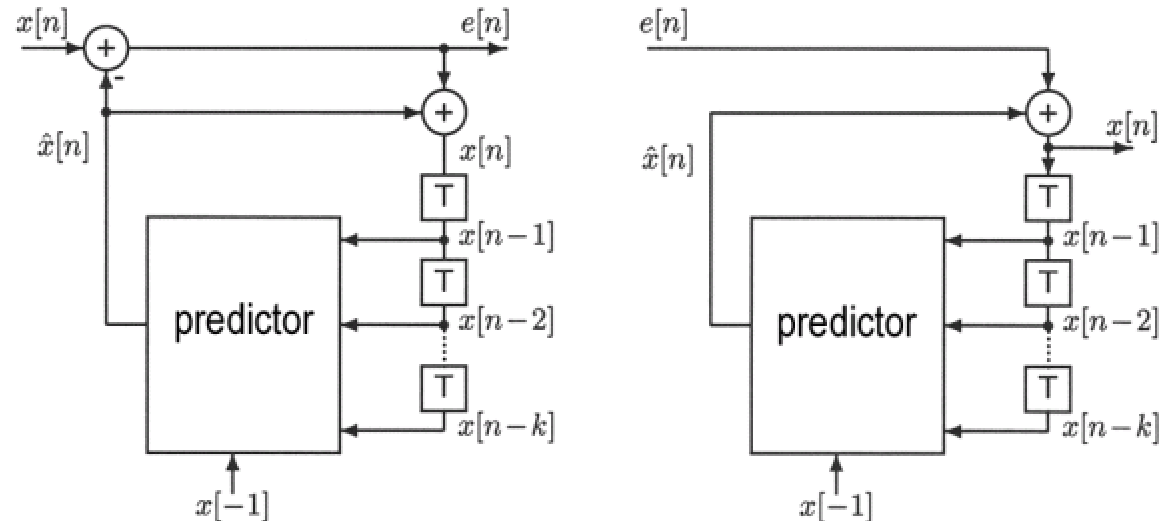
non uniform scalar quantization: pdf-optimized, perceptual optimized, snr-optimized

5.5. Differential Encoding (1)

- Belongs to predictive and pre-coding techniques
- Takes advantage of the high level of correlation between neighboring samples in sources such as speech and images by encoding differences.
- The main objective with that is a decrease of the signal variance. The smaller the signal variance is, the more concentrated the signal amplitudes upon a certain value is. This brings with it a decrease of the entropy and therefore favors a following entropy coding.
- Common application as a part of composite compression strategies, e.g. to compress low frequency components (high sample-to-sample correlation)

Differential Encoding (2)

- Block diagram: simple differential encoding system (without quantization)



with

$x[n]$, $n-th\ symbol$

$\hat{x}[n]$, $prediction\ of\ n-th\ symbol$

$e[n] = x_n - \hat{x}[n]$, $prediction\ error$

T, $time\ delay\ step$

http://www.icsy.de

Differential Encoding (3)

- Simple differential encoding system (without quantization):
  - simplest prediction: predict each symbol directly by its predecessor, i.e.

$$\hat{x}[n] = x[n-1] \quad \Rightarrow \quad e[n] = x[n] - x[n-1]$$

  - general case:

$$\hat{x}[n] = f(x[n-1], x[n-2], \ldots, x[0])$$

  - assumption: linear dependencies, i.e.

$$\hat{x}[n] = \sum_{k=1}^{p} a_k \cdot x[n-k]$$

  - optimal choice of a_k - minimization of prediction error → discrete Wiener-Hopf equations, involves autocorrelation functions

http://www.icsy.de

Differential Encoding (4)

Example: simplest prediction, quantization involved

- source sequence: 6,2 9,7 13,2 5,9 8 7,4 4,2 1,8
- difference sequence: 6,2 3,5 3,5 -7,3 2,1 -0,6 -3,2 -2,4
- quantizer with output values: -6, -4, -2, 0, 2, 4, 6
- quantized difference sequence: 6 4 4 -6 2 0 -4 -2
- reconstructed sequence: 6 10 14 8 10 10 6 4
- error between original and reconstruction:
  0,2 -0,3 -0,8 -2,1 -2 -2,6 -1,8 -2,2
- **Problem: quantization error accumulates as process continues**

http://www.icsy.de

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

ICSY Integrated Communication Systems

http://www.icsy.de

Differential Encoding (4a)

- Problem: the quantization error (here denoted by *"q"*) accumulates as the process continues
- Reason: The encoder generates a difference sequence based on original data; the decoder adds back quantized differences onto a distorted version of the original signal!

$$d_1 = x_1 - x_0$$

$$\hat{d}_1 = Q[d_1] = d_1 + q_1$$

$$\hat{x}_1 = x_0 + \hat{d}_1 = x_0 + d_1 + q_1 = x_1 + q_1$$

$$d_2 = x_2 - x_1$$

$$\hat{d}_2 = Q[d_2] = d_2 + q_2$$

$$\hat{x}_2 = \hat{x}_1 + \hat{d}_2 = x_1 + q_1 + d_2 + q_2$$

$$= x_2 + q_1 + q_2$$

•

•

•

$$\hat{x}_n = x_n + \sum_{k=1}^{n} q_k$$

Differential Encoding (4b)

- Solution: force both sides to use the same data, i.e. force the encoder to use the reconstructed sequence, too

$$d_1 = x_1 - x_0$$
$$\hat{d}_1 = Q[d_1] = d_1 + q_1$$
$$\hat{x}_1 = x_0 + \hat{d}_1 = x_0 + d_1 + q_1 = x_1 + q_1$$

$$d_2 = x_2 - \hat{x}_1$$
$$\hat{d}_2 = Q[d_2] = d_2 + q_2$$
$$\hat{x}_2 = \hat{x}_1 + \hat{d}_2 = \hat{x}_1 + d_2 + q_2$$
$$= x_2 + q_2$$

•

•

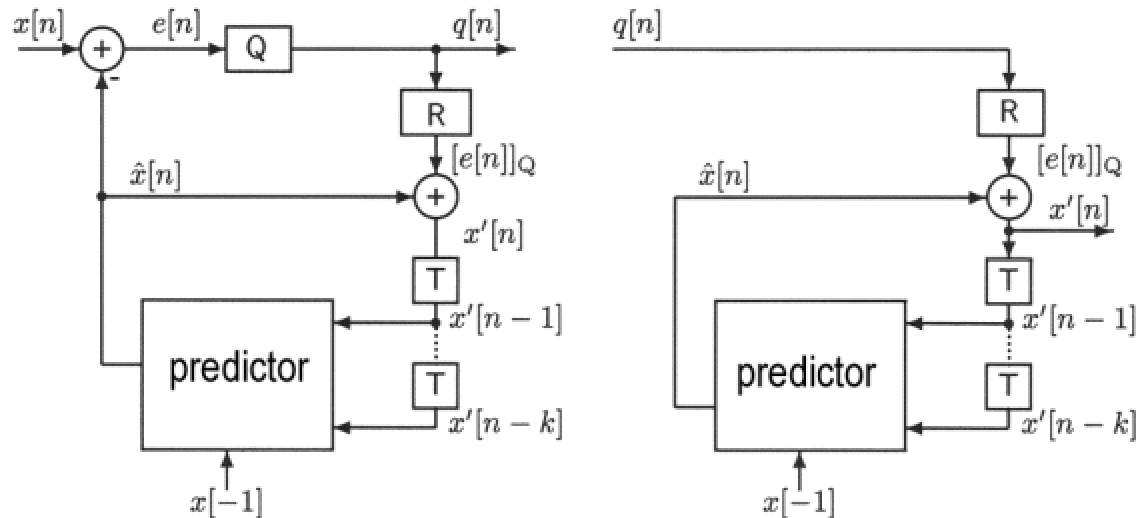
•

$$\hat{x}_n = x_n + q_n$$

http://www.icsy.de

Differential Encoding (5)

- Block diagram: simple differential encoding system (with quantization)



Differential Encoding (6)

- **DPCM** (differential pulse code modulation)
  - linear prediction dependencies, predictor coefficients by solving Wiener-Hopf equations
  - adaptive variants (predictor and quantizer)
  - invited by Bell Laboratories
  - primarily known as a speech encoding system and widely used in telephone communications
- **DM** (delta modulation)
  - a very simplified version of DPCM with a 1-Bit (two level) quantizer
  - the two level quantizer with output values +/- Δ can only represent a sample to sample difference of Δ

Differential Encoding (7)

http://www.icsy.de

- Differential encoding for two-dimensional signals:
  - general (linear) case:

$$\hat{x}[n,m]=\sum_j \sum_i a_{i,j} x[n-i,m-j] \qquad \sum_{i,j} a_{i,j}=1.0$$

  - raster scan, row by row:

$$\hat{x}=f(A,B,C,D)$$