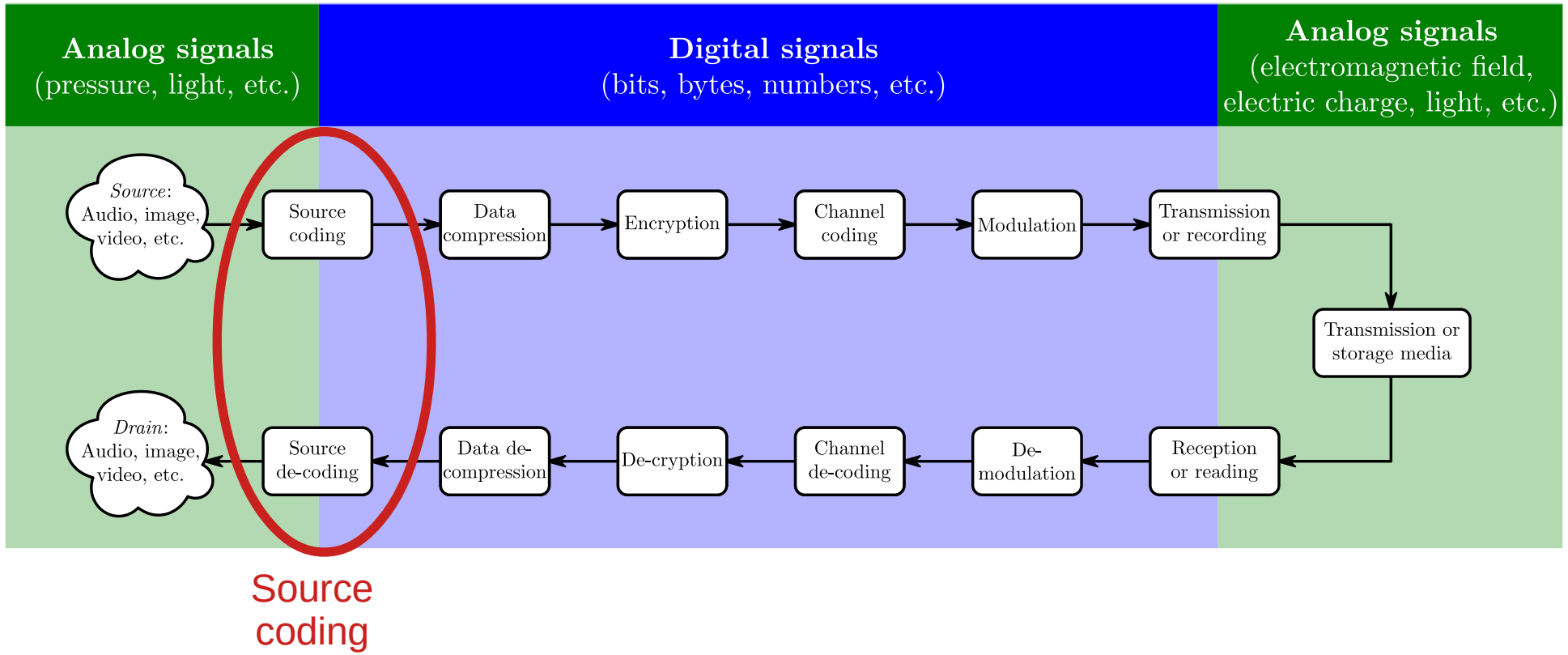
Information Transmission
Chapter 5, Source coding

OVE EDFORS
ELECTRICAL AND INFORMATION TECHNOLOGY

Learning outcomes

- After this lecture the student should
  - understand the basics of source coding,
  - know what a prefix free source code is,
  - know how to calculate average codeword length,
  - understand the limits on source coding,
  - understand the concept of universal source coding, and
  - be able to perform encoding and decoding according to the Lempel-Ziv-Welch algorithm

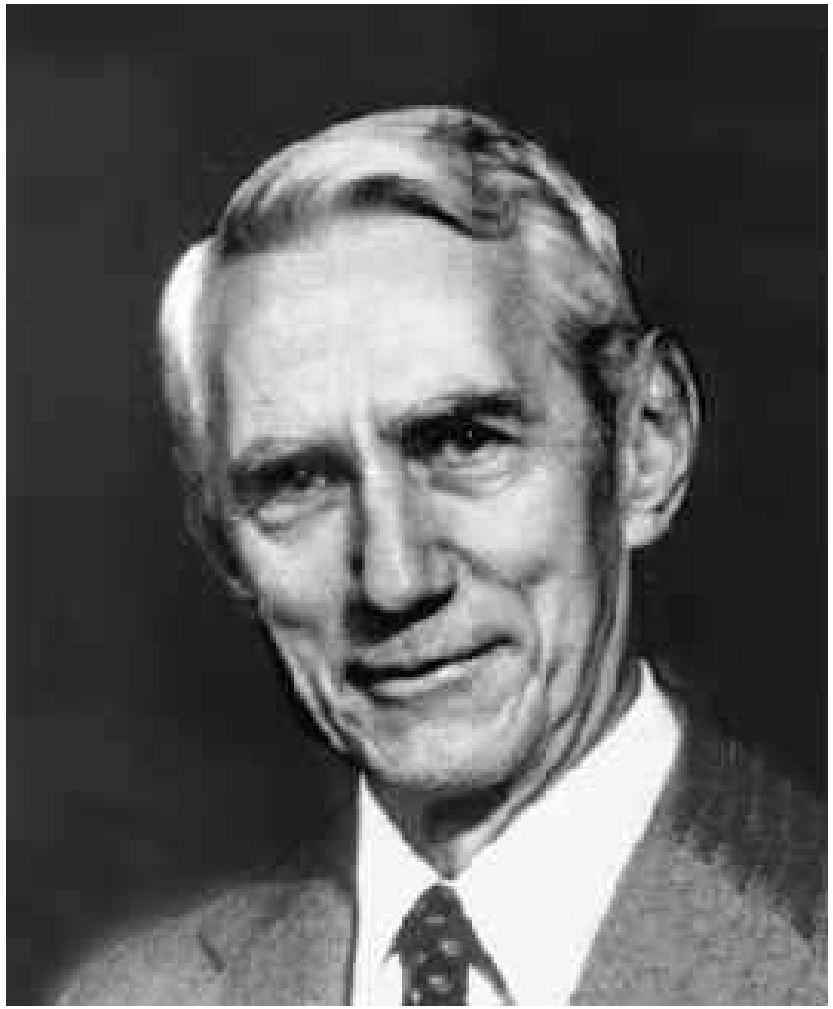
Where are we in the BIG PICTURE?



Lecture relates to pages
179-189 in textbook.

LUND UNIVERSITY

What did Shannon promise?



- We can represent a source sequence from X, of length n, uniquely by, on the average, $nH(X)$ bits

Prefix free source code

We say that a sequence of length *l* is a *prefix* of a sequence if the first *l* symbols of the latter sequence is identical to the first sequence; in particular, a sequence is a prefix of itself.

Then we require that *no codeword is the prefix of another codeword* and call such a code a *prefix-free source code*.

The sequence 10011 has the prefixes: 1, 10, 100, 1001, and 10011.

The source code with codewords {00,01,1} is prefix-free, but {00,10,1} is not, since 1 is prefix of 10.

Example

Consider the source code

u	$P_U(u)$	$\boldsymbol{x}$
u_1	0.45	0
u_2	0.30	10
u_3	0.15	110
u_4	0.10	111

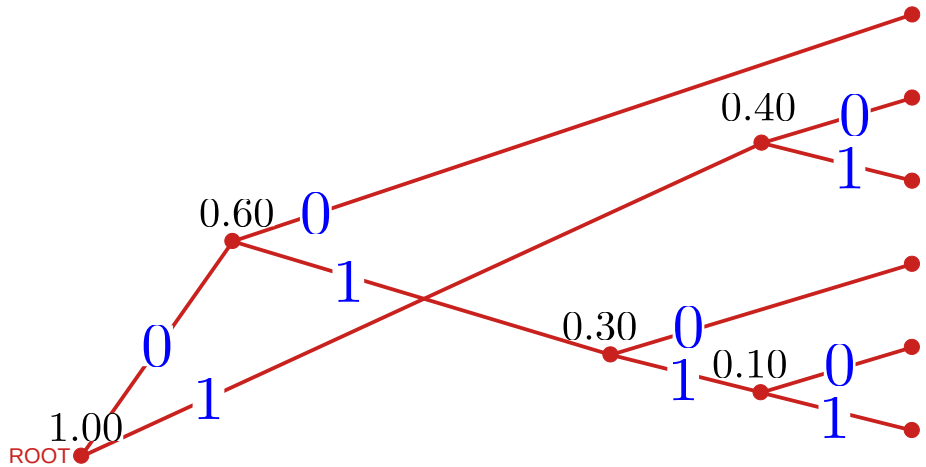
What is the average codeword length?

Huffman code construction

An optimal way to find a prefix-free variable-length code was discovered by David Huffman in 1951, when he worked on a term, paper as a student.

Procedure:

1. Let the source symbols be nodes with respective probabilities
2. Combine the two least probable remaining nodes into a new node and calculate its probability
3. If there are more than one node left, go back to step 2.
4. Label each branch of the constructed the tree either 0 or 1.
5. Traversing the tree from the root to each symbol gives the codewords.



u	$P_U(u)$	$\boldsymbol{x}$
u_1	0.30	00
u_2	0.20	10
u_3	0.20	11
u_4	0.20	010
u_5	0.05	0110
u_6	0.05	0111

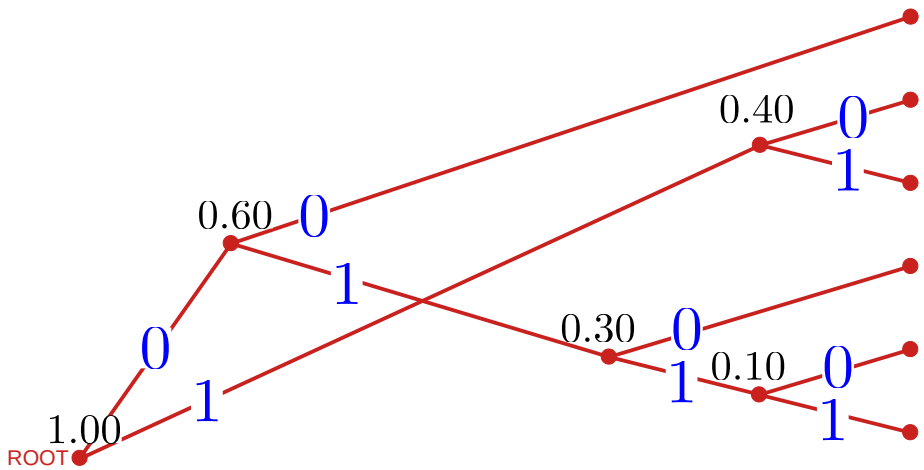
LUND UNIVERSITY

Path length lemma

In a rooted tree with probabilities, the average depth of the leaves is equal to the sum of the probabilities of the nodes (including the root).

Example

What is the average word length and the uncertainty of the source



u	$P_U(u)$	$\boldsymbol{x}$
u_1	0.30	00
u_2	0.20	10
u_3	0.20	11
u_4	0.20	010
u_5	0.05	0110
u_6	0.05	0111

Average word length: $\bar{W} = 1.00 + 0.60 + 0.30 + 0.40 + 0.10 = 2.40$

Uncertainty/entropy: $H(U) = -\sum_{i=1}^{6} P_U(u_i) \log_2(P_U(u_i)) \approx 2.35$

2% away from what is possible

LUND UNIVERSITY

Reaching the limit

If we encode consecutive source symbols pairwise, that is, use the Huffman code for the source

$u_i u_j$	$P_{U_1U_2}(u_i u_j)$
$u_1 u_1$	0.0900
$u_1 u_2$	0.0600
$u_1 u_3$	0.0600
	$\vdots$
$u_6 u_6$	0.0025

we will obtain an average codeword length per single source symbol that is closer to the uncertainty of the source, $H(U) = 2.35$.

A universal source coding algorithm

The LZW algorithm is due to Ziv, Lempel, and Welch and belongs to the class of so-called *universal source-coding algorithms* which means that we do not need to know the source statistics.

The algorithm is easy to implement and for long sequences it approaches the uncertainty of the source; it is asymptotically optimum.

Basic procedure

1. Initialize the dictionary.
2. Find the longest string W in the dictionary that matches the current input.
3. Emit the dictionary index for W to output and remove W from the input.
4. Add W followed by the next symbol in the input to the dictionary.
5. Go to Step 2.

Suppose we want to compress the sentence:

DO_NOT_TROUBLE_TROUBLE_

UNTIL_TROUBLE_TROUBLES_YOU!

DO_NOT_TROUBLE_TROUBLE_
UNTIL_TROUBLE_TROUBLES_YOU!

Step	Entry	# binary digits
0	*	—
1	D	8
2	O	$\lceil \log 1 \rceil + 8$
3	_	$\lceil \log 2 \rceil + 8$
4	N	$\lceil \log 3 \rceil + 8$
5	OT	$\lceil \log 4 \rceil$
6	T	$\lceil \log 5 \rceil + 8$
7	_T	$\lceil \log 6 \rceil$
8	TR	$\lceil \log 7 \rceil$
9	R	$\lceil \log 8 \rceil + 8$
10	OU	$\lceil \log 9 \rceil$
11	U	$\lceil \log 10 \rceil + 8$
12	B	$\lceil \log 11 \rceil + 8$
13	L	$\lceil \log 12 \rceil + 8$
14	E	$\lceil \log 13 \rceil + 8$
15	_TR	$\lceil \log 14 \rceil$
16	RO	$\lceil \log 15 \rceil$
17	OUB	$\lceil \log 16 \rceil$
18	BL	$\lceil \log 17 \rceil$
19	LE	$\lceil \log 18 \rceil$

Step	Entry	# binary digits
20	E_	$\lceil \log 19 \rceil$
21	_U	$\lceil \log 20 \rceil$
22	UN	$\lceil \log 21 \rceil$
23	NT	$\lceil \log 22 \rceil$
24	TI	$\lceil \log 23 \rceil$
25	I	$\lceil \log 24 \rceil + 8$
26	L_	$\lceil \log 25 \rceil$
27	_TRO	$\lceil \log 26 \rceil$
28	OUBL	$\lceil \log 27 \rceil$
29	LE_	$\lceil \log 28 \rceil$
30	_TROU	$\lceil \log 29 \rceil$
31	UB	$\lceil \log 30 \rceil$
32	BLE	$\lceil \log 31 \rceil$
33	ES	$\lceil \log 32 \rceil$
34	S	$\lceil \log 33 \rceil + 8$
35	_Y	$\lceil \log 34 \rceil$
36	Y	$\lceil \log 35 \rceil + 8$
37	OU!	$\lceil \log 36 \rceil$
38	!	$\lceil \log 37 \rceil + 8$

Evaluation

Without compression we need as many as 50*8 = 400 binary digits to represent the sentence as a string of 50 ASCII symbols. If we sum the number of binary digits needed for the 38 steps shown in the table we get only 271 binary digits.

A highly optimized version of the LZW algorithm we have described is used widely in practice to compress computer files on in all major operating systems.

Summary

- Most natural types of data can be compressed
- The uncertainty, or entropy, of the source determines how much we can compress data without loosing anything
- Prefix free variable-length source codes can be uniquely decoded
- Huffman's code construction is optimal for a given set of symbols
- Grouping symbols can reduce average code word length, but not further down than what is given by the uncertainty/entropy
- Universal source coding builds a coding table on the fly, without knowing the source probabilities in advance
- Lempel-Ziv-Welch is the most well known universal source coder, applied in all modern operating systems for compressing files

LUND
UNIVERSITY