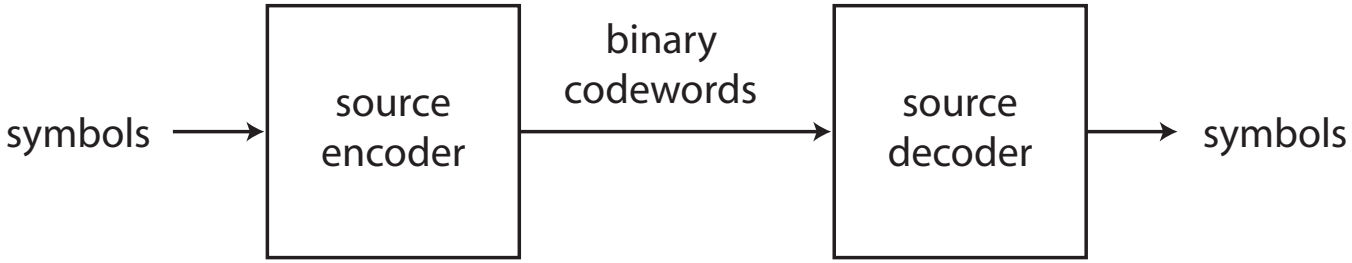
EE4304 C-term 2007: Lecture 24 Supplemental Slides

D. Richard Brown III
Worcester Polytechnic Institute, Department of Electrical and Computer Engineering

February 16, 2007

Source Encoding/Decoding



1. A "lossless" code is a unique mapping between symbols (from the discrete alphabet $\mathcal{A} = \{a_1, \ldots, a_M\}$) and binary codewords. For example, in a baseball game, the outcome of an at-bat could be encoded/decoded as

$$\begin{aligned} \text{hit} &\leftrightarrow 0 \\ \text{walk} &\leftrightarrow 10 \\ \text{out} &\leftrightarrow 11 \end{aligned}$$

2. Note that codewords do not all have to be the same length.

3. What makes a "good" lossless code?

Properties of "Good" Lossless Codes

1. Uniquely decodable. Each sequence of bits from the encoder can only be interpreted as one possible symbol sequence. Example:

   $$a_1 \leftrightarrow 1 \qquad a_2 \leftrightarrow 0 \qquad a_3 \leftrightarrow 00 \qquad a_4 \leftrightarrow 11$$

   You receive: 1011100. What symbols were sent? You can't tell if it was $a_1, a_2, a_1, a_1, a_1, \ldots$ or $a_1, a_2, a_1, a_4, \ldots$, or $a_1, a_2, a_4, a_1 \ldots$, etc. This is an example of a code that is not uniquely decodable.

2. Instantaneous. The code is constructed so that the decoder can recognize the end of a valid codework before the first bit of the next codeword is received.

3. Efficient. The average number of codeword bits per symbol (aka "average codeword length") is as small as possible. Let L_m be the number of bits in the codeword for symbol a_m. Then the average codeword length is then

   $$\bar{L} \quad = \quad \sum_{m=1}^{M} p_m L_m.$$

Note that "uniquely decodable" + "instantaneous" = "prefix code". A prefix code is a code where no codeword is a prefix for any other codeword.

Code Examples

source symbol	probability	code I	code II	code III	code IV
a_1	0.40	0	0	0	0
a_2	0.25	01	10	10	111
a_3	0.20	001	110	110	100
a_4	0.15	0001	101	111	110
uniquely decode?					
instantaneous?					
prefix code?					
$\bar{L}$					

Source Coding Theorem (Shannon, 1948)

We want our code to be efficient. Is there a limit to how small we can make our average codeword length and not lose information?

Shannon's Source Coding Theorem Given a discrete memoryless source X with entropy

$$H(X) \quad = \quad -\sum_{m=1}^{M} p_m \log_2(p_m)$$

bits of information per symbol, it is possible to encode and decode this source without loss if $\bar{L} > H(X)$ bits per symbol.

Remarks:

1. $H(X)$ is a fundamental limit on the average number of bits per source symbol needed to represent the DMS without a loss of information.
2. Shannon's source coding theorem only establishes this limit. It only says what is possible. It doesn't say how to do this.
3. Notation: $\eta = H(X)/\bar{L} =$ code efficiency. All lossless codes must satisfy $\eta < 1$.

Code Examples Revisited

source symbol	probability	code I	code II	code III	code IV
a_1	0.40	0	0	0	0
a_2	0.25	01	10	10	111
a_3	0.20	001	110	110	100
a_4	0.15	0001	101	111	110
$\bar{L}$		2.10	1.95	1.95	2.20
η					

$$H(X) \quad = \quad -\sum_{m=1}^{M} p_m \log_2(p_m) =$$

Methods for Code Generation

The two most common methods for generating prefix codes:

1. Huffman coding.
   - Easy to do with pencil and paper.
   - Harder to implement on a computer.
   - Tends to be very efficient.
   - Some loss of efficiency if the source outputs are correlated.
2. Lempel-Ziv coding.
   - Difficult to get anything interesting with pencil and paper.
   - Easy to implement on a computer.
   - Also very efficient.
   - Tends to outperform Huffman if the source outputs are correlated.

Huffman Coding Procedure

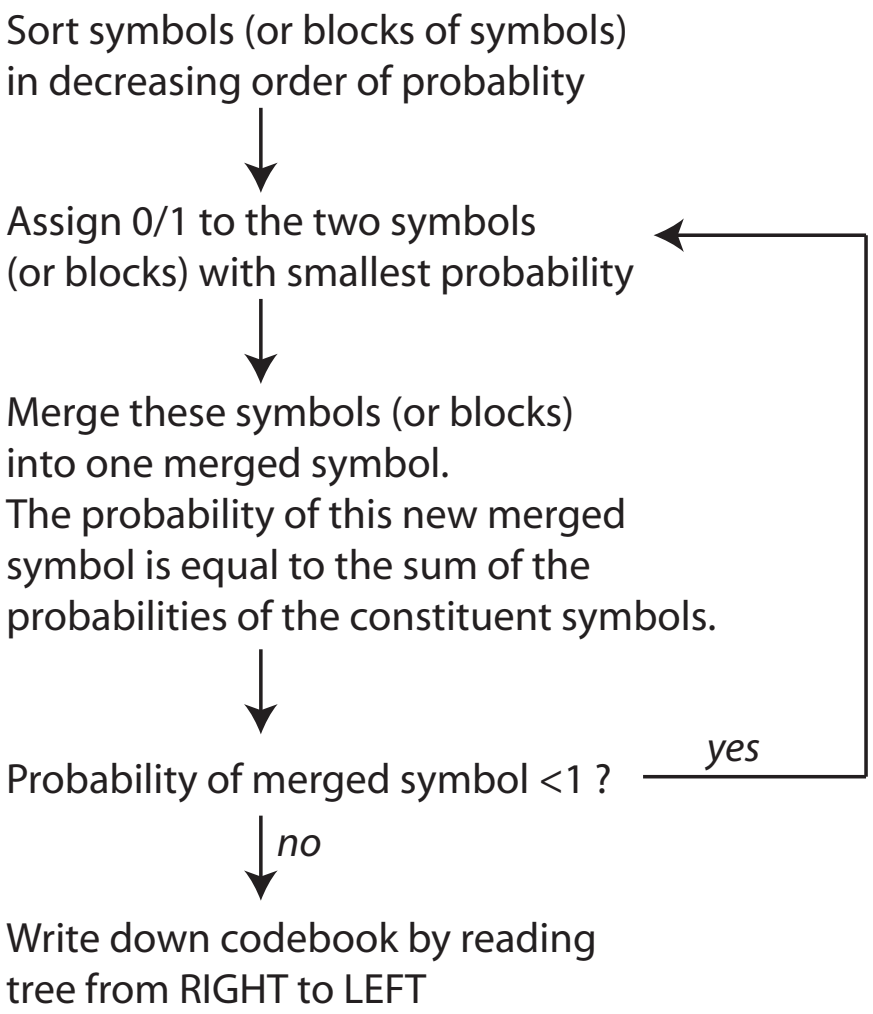
Basic idea is to develop a variable-length code that satisfies the prefix condition where

- Shorter codewords are assigned to high probability symbols (low information content)
- Longer codewords are assigned to low probability symbols (high information content)

Remarks:

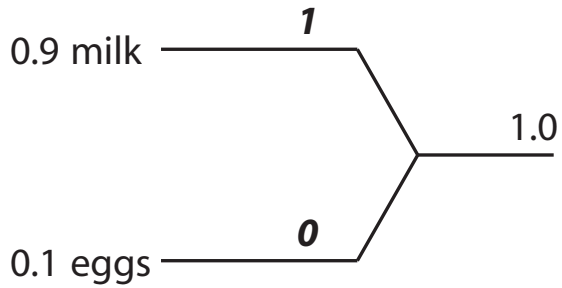
1. Need to know the DMS probabilities to apply the Huffman coding procedure.
2. The Huffman coding procedure has been shown to produce the most efficient prefix code for a DMS with known probabilities.
3. The Huffman coding procedure does not generate a unique code.
4. The Huffman coding procedure is usually applied to DMS symbols one symbol at a time but can also be applied to blocks of DMS symbols.

Huffman Coding Procedure



Huffman Coding Procedure: Example 1

Binary DMS. $\mathcal{A} = \{\text{milk,eggs}\}$. $p_1 = 0.9$, $p_2 = 0.1$.



Codebook

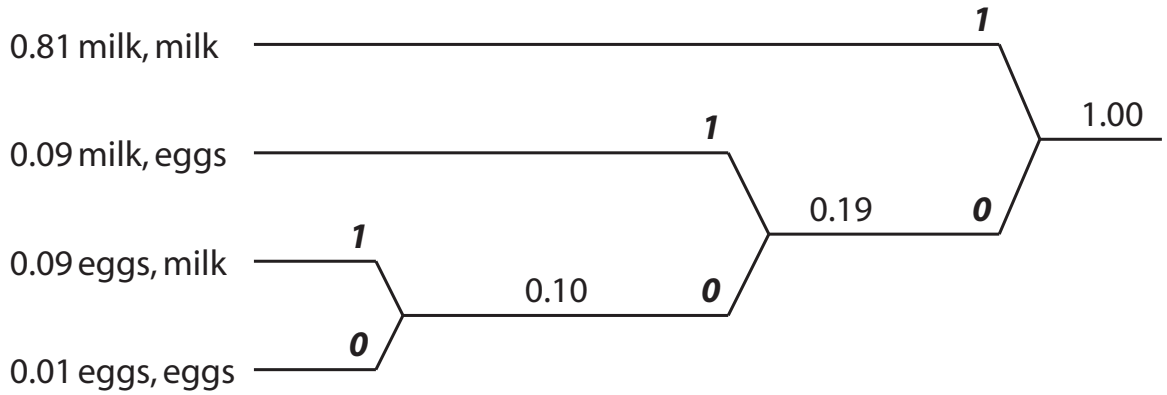
milk = 1
eggs = 0

$\overline{L} = 1$

Not very interesting...

Huffman Coding Procedure: Example 2

Same binary DMS. Let's encode *two* symbols at a time...



Codebook

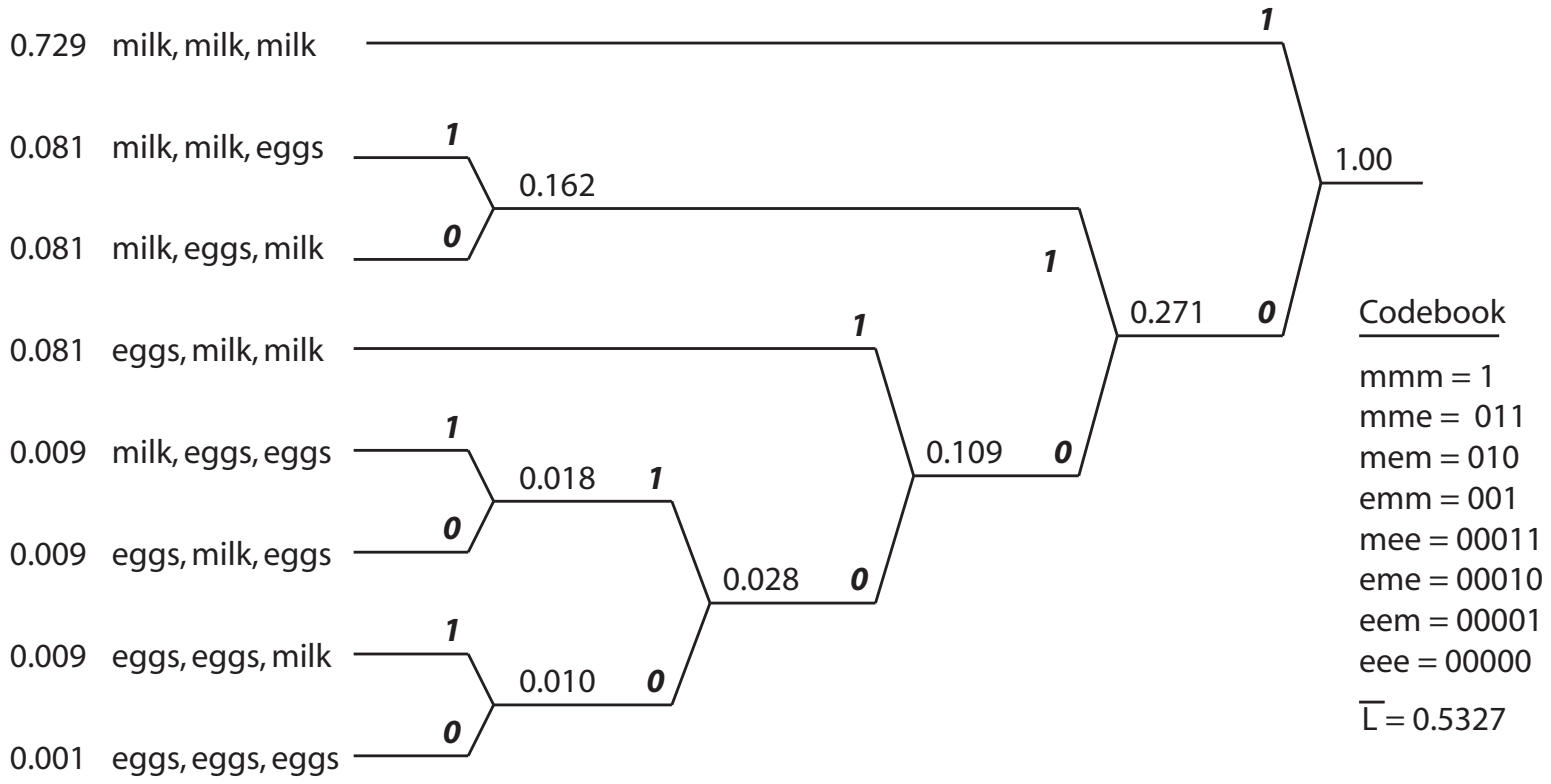
milk,milk = 1
milk,eggs = 01
eggs, milk = 001
eggs, eggs = 000

$\overline{L} = 0.645$

Big improvement!

Huffman Coding Procedure: Example 3

Same binary DMS. Let's encode *three* symbols at a time...



Even more improvement!

Final Remarks on Huffman Coding

1. Average codeword length (per symbol) will satisfy

$$H(X) \leq \bar{L} < H(X) + \frac{1}{n}$$

   where n is the block size in symbols. This result implies that you can make your code as efficient as you want, i.e. you can get as close as you want to the entropy bound, $\eta \rightarrow 1$, by increasing the symbol block size n.

2. The downside of increasing n is that the complexity and delay of the encoder/decoder increases.