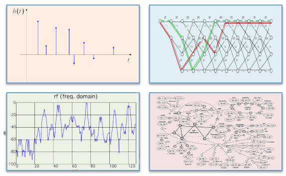
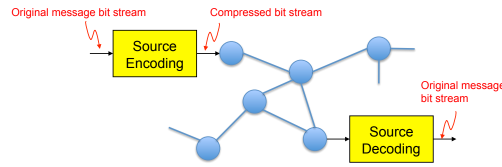


INTRODUCTION TO EECS II
DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2010
Lecture #22: Lossless Source Coding
November 29, 2010

- Information & entropy
- Variable-length codes: Huffman's algorithm
- Adaptive codes: LZW algorithm

Efficiency via Source Coding



An example of an end-to-end application-layer protocol – it doesn't involve intermediate nodes in the network.

Idea: Many message streams use a "natural" fixed-length encoding: 7-bit ASCII characters, 8-bit audio samples, 24-bit color pixels. But if we're willing to use variable-length encodings (message symbols of differing lengths) we could assign short encodings to common symbols and longer encodings to other symbols… this should shorten the average length of a message.

Measuring Information Content

Suppose you're faced with N equally probable choices, and I give you a fact that narrows it down to M choices. Claude Shannon offered the following formula for the information you've received:

$\log_2(N/M)$ bits of information

Information is measured in bits (binary digits), which you can interpret as the number of binary digits required to encode the choice(s)

Examples:
- information in one coin flip: $\log_2(2/1) = 1$ bit
- roll of 2 dice: $\log_2(36/1) = 5.2$ bits
- outcome of a Patriots game: 1 bit
  (well, actually, are both outcomes equally probable?)

When choices aren't equally probable

When the choices have different probabilities (p_i), you get more information when learning of a unlikely choice than when learning of a likely choice. Shannon defined:

Information from choice$_i$ = $\log_2(1/p_i)$ bits

We can use this to compute the information content taking into account all possible choices:

Expected information content in a choice = $\Sigma p_i \cdot \log_2(1/p_i)$

This characterization of the information content in learning of a choice is called the *information entropy* or *Shannon's entropy*.

Goal: Match Data Rate to Info Rate

- Ideally we want to find a way to encode message so that the transmission data rate would match the information content of the message
- It can be hard to come up with such a code!
  - Transmit results of 1000 flips of unfair coin: p(heads) = p_H
  - Avg. info in unfair coin flip:
    $p_H \log_2(1/p_H) + (1-p_H) \log_2(1/(1-p_H))$
  - For $p_H = .999$, this evaluates to .0114
  - Goal: encode 1000 flips in 11.4 bits! How?
  - What's the code? Hint: can't encode each flip separately
- Conclusions
  - Effective codes leverage context
    - How to encode Shakespeare sonnets using just 8 bits?
    - What if some sonnets are more popular than others?
  - Effective codes encode sequences, not single symbols

Example

$choice_i$	p_i	$\log_2(1/p_i)$
"A"	1/3	1.58 bits
"B"	1/2	1 bit
"C"	1/12	3.58 bits
"D"	1/12	3.58 bits

Expected information content in a choice
= (.333)(1.58) + (.5)(1) + (2)(.083)(3.58)
= 1.626 bits

Can we find an encoding where transmitting 1000 choices requires 1626 bits on the average?

The naïve fixed-length encoding uses two bits for each choice, so transmitting the results of 1000 choices requires 2000 bits.

Variable-length Encodings of Symbols

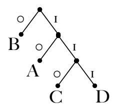
(David Huffman, MIT 1950)

Use shorter bit sequences for high probability choices, longer sequences for less probable choices

choice$_i$	p_i	*encoding*
"A"	1/3	10
"B"	1/2	0
"C"	1/12	110
"D"	1/12	111

BC A BA D

011010010111



Huffman Decoding Tree

Expected length
=(.333)(2)+(.5)(1)+(2)(.083)(3)
= 1.666 bits

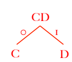
Transmitting 1000 choices takes an average of 1666 bits… good, but not optimal (1626 bits)

Huffman's Coding Algorithm

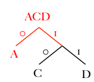
- Begin with the set S of symbols to be encoded as binary strings, together with the probability p(s) for each symbol s in S. The probabilities sum to 1.
  - In the example from the previous slide, the initial set S contains the four symbols and their associated probabilities from the table.
- Repeat these steps until there is only 1 symbol left in S:
  - Choose the two members of S having lowest probabilities. (Resolve ties arbitrarily.)
  - Remove the selected symbols from S, and create a new node of the decoding tree whose children (sub-nodes) are the symbols you've removed. Label the left branch with a "0", and the right branch with a "1" (you can also do it the other way around).
  - Add to S a new symbol that represents this new node. Assign this new symbol a probability equal to the sum of the probabilities of the two nodes it replaces.

Huffman Coding Example

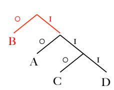
- Initially S = { (A, 1/3) (B, 1/2) (C, 1/12) (D, 1/12) }
- First iteration
  - Symbols in S with lowest probabilities: C and D
  - Create new node
  - Add new symbol to S = { (A, 1/3) (B, 1/2) (CD, 1/6) }



- Second iteration
  - Symbols in S with lowest probabilities: A and CD
  - Create new node
  - Add new symbol to S = { (B, 1/2) (ACD, 1/2) }



- Third iteration
  - Symbols in S with lowest probabilities: B and ACD
  - Create new node
  - Add new symbol to S = { (BACD, 1) }



Huffman Codes - The Final Word?

- Given static symbol probabilities, the Huffman algorithm creates an optimal encoding when each symbol is encoded separately
- Huffman codes have the biggest reduction in average message length when some symbols are substantially more likely than other symbols
- You can improve the results by adding encodings for symbol pairs, triples, quads, etc.
  - But the number of possible encodings quickly becomes intractable
  - *Pairs: 1.646 bits/sym, Triples: 1.637, Quads 1.633, …*
- To get a more efficient encoding (closer to information content) we need to encode sequences of choices, not just each choice individually
- Symbol probabilities change message-to-message, or even within a single message
- Can we do adaptive variable-length encoding?
- Yes – the LZW algorithm does just that!

Summary

- Source coding: recode message stream to remove redundant information, aka compression.
- Our goal: match data rate to actual information content.
- Information content from choice$_i$ = $\log_2(1/p_i)$ bits
- Shannon's Entropy: average information content on learning a choice = $\Sigma p_i \cdot \log_2(1/p_i)$.
- Huffman's encoding algorithm builds optimal variable-length codes when symbols encoded individually.