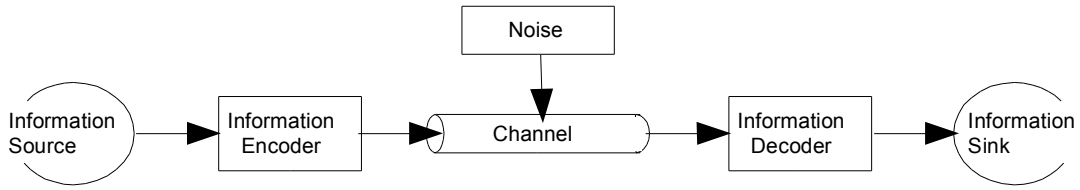
Information Transmission and Coding

Department of Computer Science
COMPSCI 314 S1 C 2006

Introduction

We deal here with the transmission of information through a channel, as shown in the figure, with the attendant coding of information to allow transmission.



The important terms here are –

- **Information** is anything that has meaning, especially something that is somewhat unexpected.
- The **Information Source** is some thing that generates information
- The **Information Encoder** converts the information into a form that can be handled by the rest of the system
- The **Channel** carries the encoded information from one place to another (data transmission) or from from one time to another (data storage and retrieval)
- The **Information decoder** recovers the encoded information (reverses the encoder)
- The **Information Sink** receives or processes the decoded or recovered information
- **Noise** is anything that corrupts the information being sent through the Channel
- The **Capacity** is the amount of information that can be delivered by the channel to decoder. For a *noiseless channel* (with no noise input) it is usually identical to the bit rate, but for a *noisy channel* will be less.

Important matters considered by Information and Coding Theory are –

1. How can we code information to get the maximum possible amount through a noiseless channel? This leads particularly to data compression, both *lossless* or *text* compression where the output must be bitwise identical to the original and *lossy* compression, where the result just has to look or sound the same to a person.
2. How can we minimise the effect of noise on transmission through a noisy channel?

Noiseless Coding

In general we take a *symbol* from the information source and convert it into a *codeword* for transmission over the channel. The channel can deliver only so many bits per second, so to maximise the symbol transmission rate we want to maximise the number of symbols per bit or, more usually minimise the bits/symbol. It should be obvious that frequent symbols should be represented by the shorter codewords. Questions are – what are the limits, and how do we approach them?

The information "Source" *S* emits symbols chosen from an alphabet of *k* symbols $\{S_0, S_1, \ldots S_{k-1}\}$ (say the letters a…z, A…Z , the digits 0…9 plus punctuation) occurring with probabilities $\{P_0, P_1, \ldots P_{k-1}\}$. Receiving a symbol S_i gives a quantity of information $\log(1/P_i)$ which is approximately the "surprise" in receiving the symbol. The average information per symbol is known as the Entropy of the source (*H*) and is given by the following formula. Note that *H* tends to 0 both as $P \to 0$ (improbable symbol) and also as $P \to 1$ (expected symbol contains little information)

$$H = \sum_{i=0}^{k-1} P_i \log_2\left(\frac{1}{P_i}\right) = -\sum_{i=0}^{k-1} P_i \log_2(P_i)$$

If each symbol S_i is encoded into a codeword of length L_i then the average codeword length per symbol is $L = \sum_{i=0}^{k-1} P_i L_i$

A fundamental result is that $L \geq H$ (average codeword length and entropy per symbol), with equality if and only if $P_i = -\log_2 L_i$ for all *i*.

In practice, we must encode the most probable symbols with the shortest codes if possible, so that each length is $L_i = -\log_2 P_i$

The **Huffman Code** is a practical method of constructing such a code, given the symbol probabilities.

(The much more complex "arithmetic code" is often better because it is not limited to whole bits per codeword; the fractional bits of the logarithmic length can "roll over" into the next codeword.)

We also find that the efficiency of a code improves by coding an extension of the code (encoding several symbols together). This is shown in the Huffman code examples at the end for the code {A, B}, with $P_A = ¾$ and $P_B = ¼$.

Lossless Compression

The simplest lossless compressor just codes symbols according to their probabilities, as with a Huffman code, and often with a "dynamic" Huffman code that adjusts its symbol probabilities and coding as it proceeds (achieving say about 4 bits/char for most text). Many compressors build a dictionary of known "phrases" and emit pointers or indices into the phrase dictionary, such as the two described by Ziv & Lempel –

- **LZ-77** is used by most computer compressors, such as GZIP. It uses the most recent 8 – 32 k bytes of the file as its dictionary and emits {displacement, length} pairs back from the current position (or a single byte if there is no match). Decompression is just a matter of copying phrases from the given position in the window of recent text.
- **LZ-78** and its variant known as **LZW** (and often just as Ziv-Lempel) uses an explicit dictionary of usually about 4000 phrases, initialised to just the 256 ASCII characters. At each stage it examines the unprocessed input text and emits the index of the longest matching phrase, at worst just one byte. It then builds a new phrase by joining the preceding phrase and the first symbol of the one just matched. The dictionary thus expands as it accumulates phrases from the text; it is often purged when it becomes full. This is the usual compressor with data communications as it can be made very fast, but gives less compression than the LZ-77 derivatives. Decompression involves reading a phrase from the dictionary (identified by the index just read) and then creating a new one by adding to the previous phrase.

More complex compressors restrict the range of possible symbols at each stage according to their preceding "contexts" and often achieve less than 2 bits/char on text, but are seldom used in data communications. But compressors such as BZIP are widely used in program distribution.

Continuous or Analogue Transmission

The previous notes have all assumed digital, or noiseless transmission over a digital channel. But ultimately most transmission is over an analogue channel of some bandwidth W and with some noise. The question is "What is the Channel Capacity?", or "How many bits/second can the channel carry?"

The Channel Capacity C was investigated by Shannon in 1948. Start with two observations –

1. If the bandwidth W doubles, we expect to be able to get twice as many signal changes in a given time, so $C \propto W$.
2. The important aspect for noise is the ratio of signal power S to noise power N (always appearing as the "signal-noise ratio" S/N). For a given S/N we can expect to distinguish some number of different signal levels; if S/N doubles we can expect to decide twice as many states, or 1 more bit. Therefore $C \propto \log_2(S/N)$.
   [The S/N ratio is usually measured in deciBels (symbol dB), defined as $S/N_{dB}=10\log_{10}(S/N)$.]

In fact Shannon showed (with n-dimensional geometry) that

$$C = W log_2\left(1+\frac{S}{N}\right)$$

This is "Shannon's theorem" and is fundamental to all modern information transmission, setting a bound to capacity for given bandwidth and noise.

Error Correcting Codes

- The simplest are the **Block codes**, such as Hamming, where a fixed-size block of data has an ECC code applied to it; corrections for one block are quite independent of other blocks.
- **Convolutional Codes** have an encoding mechanism "sliding" along the information, interspersing parity bits at frequent intervals. Decoding is usually by a Viterbi or Trellis decoder searching through the bit patterns that are compatible with data and the known coding.
- **Turbo Codes** are a recent development with two parallel encoders, one with permuted data; the decoders exchange information and mutually assist each other in resolving errors.

When decoded most Error Correcting Codes generate a "syndrome" which is usually zero if there is no error and otherwise shows the position of the error(s) and how to correct.