Fundamentals of Media Compression

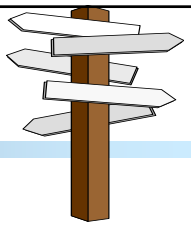
Prof. Ebroul Izquierdo

School of Electronic Engineering and Computer Science
Queen Mary, University of London

Seminario sobre TV Digital

Universidad del Valle
Cali – Colombia, October 28th 2008

Road Map

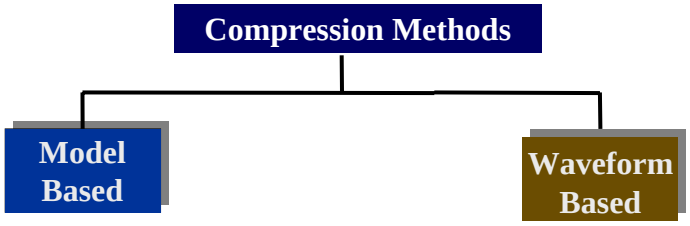


- **Fundamentals of Compression**
- **Basic Algorithms**
  - **Shannon-Fano**
  - **Huffman**
  - **Arithmetic**
  - **Dictionary Based**
- **Decorrelation Transfoms**

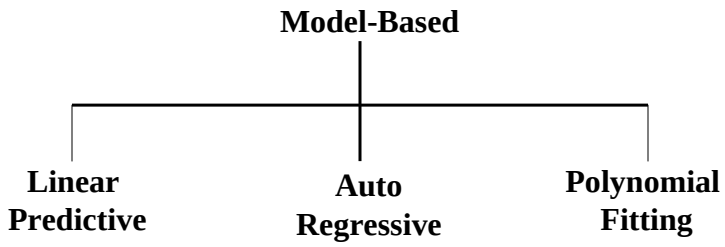
Basics of Compression

Compression reduces signal size by taking advantage of correlation

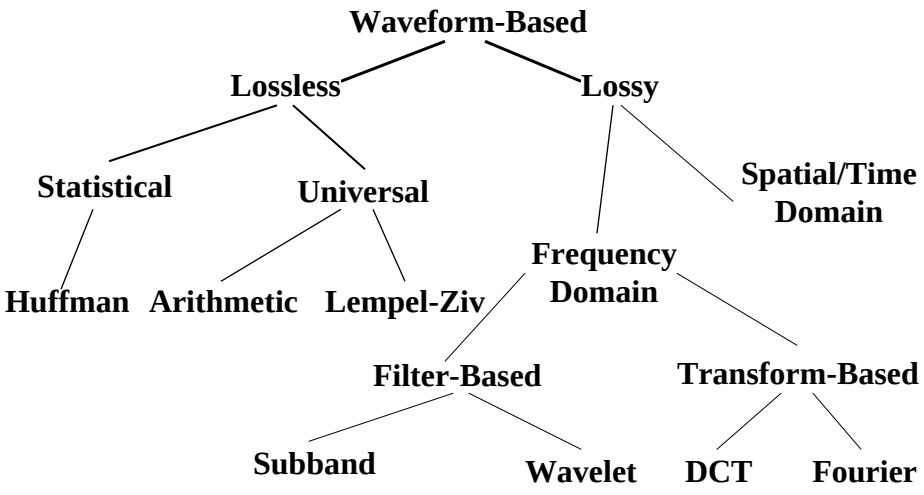
- Spatial
- Temporal
- Spectral



Model-Based Compression



Waveform-Based Compression



Compression Features

Lossless compression

- Coding Efficiency
  - Compression ratio
- Coder Complexity
  - Memory needs
  - Power needs
  - Operations per second
- Coding Delay
- **Scalability**

Compression: Theoretical Background

Run-length encoding

Average Information Entropy

For source S generating symbols S_1 through S_N

- Self entropy: $I(s_i) = \log \frac{1}{p_i} = -\log p_i$
- Entropy of source S: $H(S) = \sum_i p_i \log_2 p_i$
- Average number of bits to encode ≤ H(S) - Shannon

Differential Encoding: to improve the probability distribution of symbols

Information Theory

Shannon defined the **entropy** of an information source S as:

$$H(S) = \Sigma_i (p_i \log_2 (1/p_i))$$

- $\log_2 (1/p_i)$ indicates the amount of information contained in S_i, i.e., the number of bits needed to code S_i.
- For example, in an image with uniform distribution of gray-level intensity, i.e. $p_i = 1/256$, the number of bits needed to code each gray level is 8 bits. The entropy of this image is 8.

Information Theory (II)

- Entropy is a measure of how much information is encoded in a message. The higher the entropy,the higher the information content.
- The unit of entropy is *bits per symbol*.
- Entropy gives the actual number of bits of info. contained in a message.
- If the probability of the character ‘e’ appearing in this slide is 1/16, for example. The information content of the character is 4 bits. So, the character string “eeeee” has a total content of 20 bits (contrast this to 8-bit ASCII coding that could result in 40 bits).

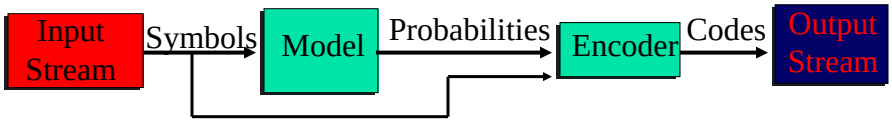
Data Compression (I)

Data Compression consists of taking a stream of symbols and transforming them into codes.

- The decision to output a certain code for a certain symbol (or set of symbols) is *based* on the *model*.
- A coder uses a model to spit out codes when its given input symbols

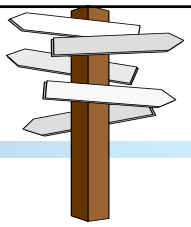
Data Compression (II)

Let's take Huffman coding to demonstrate the distinction:



- The output of the encoder is determined by the probabilities fed by the Model. *Higher prob. shorter code.*
- Model A could determine raw prob. of each symbol occurring anywhere in the i/p stream. (p_i = *# of occurrences of* S_i */ Total number of Symbols*)
- Model B could determine prob. based on the last 10 symbols in the i/p stream. (*continuously re-computes the probabilities*)

Road Map



- **Fundamentals of Compression**
- **Basic Algorithms**
  - **Shannon-Fano**
  - **Huffman**
  - **Arithmetic**
  - **Dictionary Based**
- **Decorrelation Transforms**

The Shannon-Fano Algorithm

- Calculate the frequencies of the list of symbols (organize as a list).
- Sort the list in (decreasing) order of frequencies.
- Divide list into two halfs, with the total freq. Counts of each half being as close as possible to the other.
- The upper half is assigned a code of 0 and lower a code of 1.
- Recursively apply steps 3 and 4 to each of the halves, until each symbol has become a corresponding code leaf on a tree.

Huffman Encoding (I)

Let an alphabet have N symbols $S_1 \ldots S_N$

Let p_i be the probability of occurrence of S_i

Order the symbols by their probabilities

$p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_N$

Replace symbols S_{N-1} and S_N by a new symbol H_{N-1} such that it has the probability $p_{N-1} + p_N$

Repeat until there is only one symbol

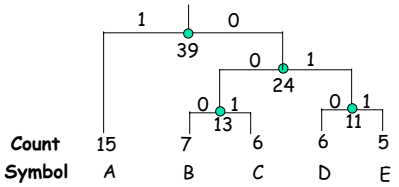
This generates a binary tree

Huffman Encoding (II)

Initialization: Put all nodes in an OPEN list *L*, keep it sorted at all times (e.g., ABCDE).

Repeat the following steps until the list *L* has only one node left:

1. **From *L* pick two nodes having the lowest frequencies, create a parent node of them.**
2. **Assign the sum of the children's frequencies to the parent node and insert it into OPEN.**
3. **Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.**



Huffman Encoding (II)

Symbol	Count	Info. $-\log_2(p_i)$	Code	Subtotal# of Bits
A	15	1.38	1	15
B	7	2.48	000	21
C	6	2.70	001	18
D	6	2.70	010	18
E	5	2.96	011	15

85.25

87

Properties of Huffman Codes

Fixed-length inputs become variable-length outputs

Average codeword length: $\sum l_i p_i$

Upper bound of average length: $\boldsymbol{H(S) + p_{max}}$

Code construction complexity: ***O(N log N)***

Prefix-condition code: no codeword is a prefix for another

- Makes the code uniquely decodable

Huffman Decoding (I)

Look-up table-based decoding

Create a look-up table

- Let L be the longest codeword
- Let c_i be the codeword corresponding to symbol s_i
- If c_i has l_i bits, make an L-bit address such that the first l_i bits are c_i and the rest can be any combination of 0s and 1s.
- Make 2^(L-l_i) such addresses for each symbol
- At each entry, record, (s_i, l_i) pairs

Huffman Decoding (II)

Decoding

- Read L bits in a buffer
- Get symbol s_k, that has a length of l_k
- Discard l_k bits and fill up the L-bit buffer

Arithmetic Coding (I)

Arithmetic coding is based on the concept of *interval subdividing*.

- In arithmetic coding a source ensemble is represented by an interval between 0 and 1 on the real number line.
- Each symbol of the ensemble narrows this interval.
- As the interval becomes smaller, the number of bits needed to specify it grows.
- Arithmetic coding assumes an explicit probabilistic model of the source.
- It uses the probabilities of the source messages to successively narrow the interval used to represent the ensemble. A high probability message narrows the interval less than a low probability message, so that high probability messages contribute fewer bits to the coded ensemble

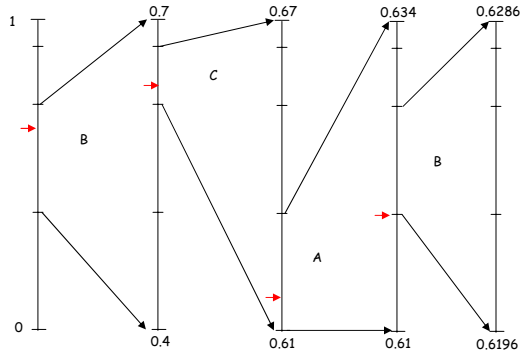
Arithmetic Coding (II)

- Now we begin with the interval [0,1) and subdivide the interval iteratively.
- For each symbol entered, the current interval is divided according to the probabilities of the alphabet.
- The interval corresponding to the symbol is picked as the interval to be further proceeded.
- The procedure continues until all symbols in the message have been processed.
- Since each symbol's interval does not overlap with others, for each possible message there is a unique interval assigned.
- We can represent the message with the interval's two ends [L,H). In fact taking any single value in the interval as the encoded code is enough, and usually the left end L is selected.

Arithmetic Coding (III)

When decoding the code v is placed on the current code interval to find the symbol x so that Q[x-1] <= code < Q[x]. The procedure iterates until all symbols are decoded.

Symbol	Probability	Q[x-1], Q[x]
A	0.4	0.0, 0.4
B	0.3	0.4, 0.7
C	0.2	0.7, 0.9
D	0.1	0.9, 1.0



String: BCAB

Code sent: 0.6196

Dictionary-Based Compression (I)

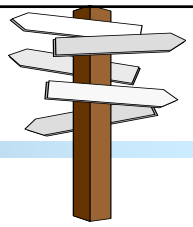
The compression algorithms we studied so far use a statistical model to encode single symbols

- Compression: Encode symbols into bit strings that use fewer bits.

Dictionary-based algorithms do not encode single symbols as variable-length bit strings; they encode variable-length strings of symbols as single tokens

- The tokens form an index into a phrase dictionary
- If the tokens are smaller than the phrases they replace, compression occurs.

Road Map



- **Fundamentals of Compression**
- **Basic Algorithms**
  - **Shannon-Fano**
  - **Huffman**
  - **Arithmetic**
  - **Dictionary Based**
- **Decorrelation Transforms**

Fourier Approximation

In mathematical notation what Fourier proved was that for any continuous function f the following relation is satisfied:

$$f(x) = \sum_{k=0}^{\infty} ak \cos\left(\frac{2\pi kx}{L}\right) + bk \sin\left(\frac{2\pi kx}{L}\right)$$

$$= a0 + \sum_{k=1}^{\infty} ak \cos\left(\frac{2\pi kx}{L}\right) + bk \sin\left(\frac{2\pi kx}{L}\right)$$

Discrete Cosine Transform (DCT)

DCT is a mathematical process related to Fourier Transform. It changes spatial intensity values to spatial frequency values.

- roughly arranges values from lowest frequency to highest frequency:
  - lowest frequencies represent coarse details
  - highest frequencies represent fine detail
  - some high frequency parts can be dropped!

DCT exploits features of the human eye:

- the eye is unable to perceive brightness levels above or below certain thresholds
- gentle gradations of brightness of colour are more important to the eye than abrupt changes.

Discrete Cosine Transform

The one-dimensional transform is defined by:

$$t(l) = c(l) \sum_{k=0}^{N-1} I(k) \cos\left(\frac{\pi(2k+1)}{2N} l\right)$$

where I is the array of N original values (the luminance values of the pixel matrix), t is the array of N transformed values (the frequency coefficients), and the c is the scaling factor for the coefficients given by

$$c(0) = \sqrt{1/N} \qquad c(l) = \sqrt{2/N}$$

Discrete Cosine Transform

- The DCT is separable: in order to estimate the DCT coefficients of a two-dimensional signal for an image

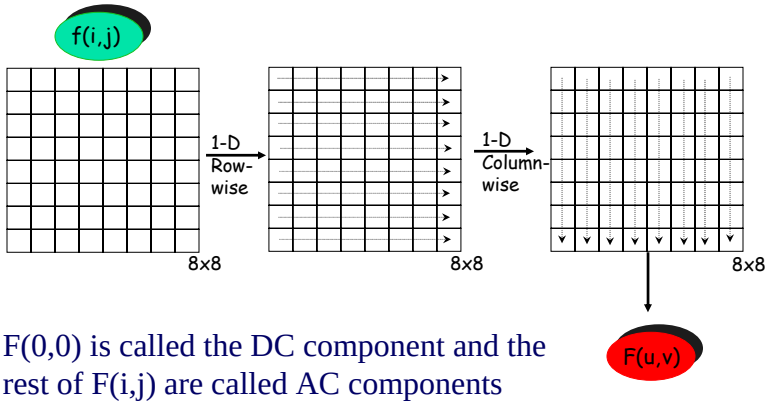
  $I(x, y)$, the one-dimensional transform can be performed twice: once for the rows, the y-axis, and once for the columns, the x-axis, of the image.
- The DCT is invertible: therefore it allows us to move back and forth between spatial and frequency domains

The one-dimensional inverse DCT is given by:

$$I(k) = \sum_{l=0}^{N-1} c(l)t(l) \cos\frac{\pi(2k+1)l}{2N}$$

Computing a 2D DCT (for Images)

Two series of 1-D transforms result in a 2-D transform as demonstrated in the figure below



F(0,0) is called the DC component and the rest of F(i,j) are called AC components