Definition

What is compression?

The minimization of the number of bits in representing an image or a video, while

- Being able to exactly reconstruct the original data (**lossless compression**)

or

- Maintaining an acceptable quality of the reconstructed original data (**lossy compression**)

Why Compress?

Telephony (220-3400 Hz)	8000 samples/s x 12 b/sample = 96 kbps
Wideband speech (50-7000 Hz)	16,000 samples/s x 14 b/sample = 224 kbps
Wideband audio (20-20,000 Hz)	44,100 samples/s x 2 channels x 16 b/sample = 1.412 Mbps
Color images	512 x 512 pixels x 24 bpp = 6.3 Mbits = 786 Kbytes
Video (CCIR601)	720 x 480 gray pixels x 8 bpp x 30 frames/s + 2 x 360 x 480 chroma pixels x 8 bpp x 30 frames/s = 166 Mbps
HDTV	1280 x 720 color pixels x 24 bpp x 60 frames/s = 1.3 Gbps

CCIR601: Without compression we could store

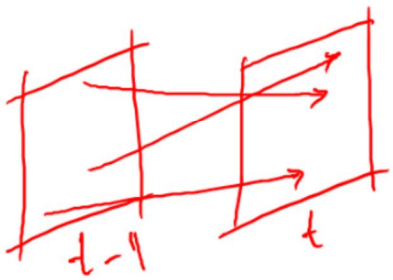
- 31 s of video on a CD-ROM (650 Mbytes)
- 4 minutes of video on a DVD-5 (40 Gbits)

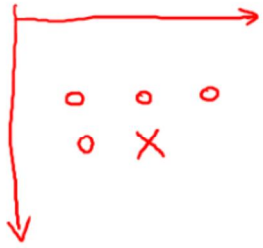
HDTV Terrestrial Broadcasting:

- 19.3 Mbits/s (6 MHz channel)
- need compression ratio ~ 70

Why are Signals Compressible?

- Statistical redundancy or structure in the data (spatial, temporal, spectral)
- Existence of perceptually irrelevant information





Compression is one of the
enabling technologies
behind the multimedia revolution
we are experiencing

Proliferation of Compression Applications

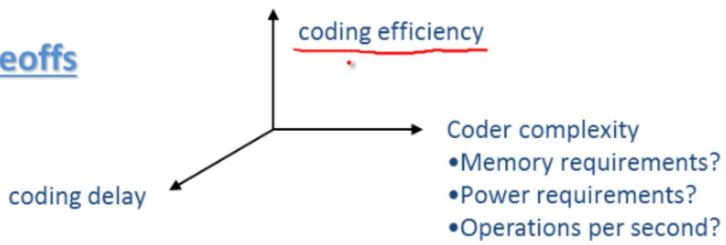
- Over 50 years of extraordinary theoretical results on compression
- Use of perceptually based distortion measures
- Successful standards
- Unparalleled DSP capabilities

- Technological advances in computers, networks, and telecommunications

Lossless Compression

- **OBJECTIVE**: *Exact reconstruction of original data from compressed data, i.e., a **reversible** process*
- **Compression ratio** (input size over output size) is determined by the **entropy** of the source
- Low Compression ratios
- Application examples: text, medical images, **lossy codecs**

tradeoffs



Lossless Source Coding

Statistical Methods
(source statistics known)

- Huffman
- Extended Huffman
- Other (Gilbert, Fano)

Applications
- Group 3 FAX
- Group 4 FAX

Universal Methods
(source statistics unknown)

- Arithmetic Coding
- Dictionary Techniques (LZxx, LZW)
- Other (adaptive Huffman)

Applications
- JBIG (arithmetic)
- Unix Compress, GIF, V.42 bis (LZW)

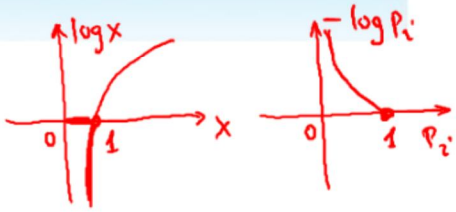
Elements from Information Theory

- Any information generating process can be viewed as a **source** that emits a sequence of symbols **randomly** chosen from a finite alphabet
  - Examples: natural written language; n-bit images (2^n symbols)
- Simplest case: **discrete memoryless source** (DMS) – successive symbols produced by the source are statistically independent (and identically distributed)

Entropy

- **Self Information:** $I(s_i) = \log \frac{1}{p_i} = -\log p_i$

Base of log: 2 → unit: bits; e → nats; 10 → Hartleys



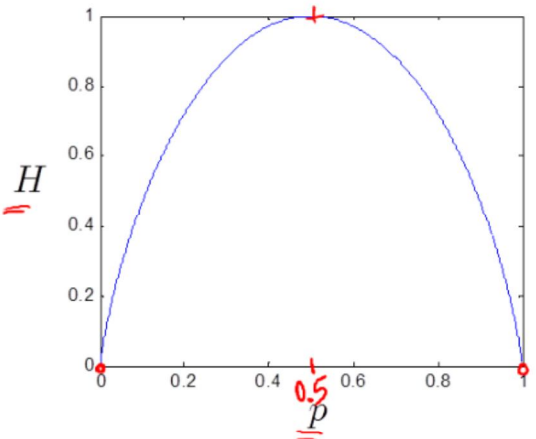
  - The occurrence of a less probable event provides more information
  - The information of independent events taken as a single event equals the sum of the information

$$I(s_1 s_2) = \log \frac{1}{p(s_1 s_2)} = \log \frac{1}{p(s_1)p(s_2)} = -\log\left(p(s_1)p(s_2)\right) = -\log p_1 - \log p_2 = I(s_1) + I(s_2)$$

- **Entropy** of a DMS: **Average information** per symbol

$$H(S) = \sum_{i=1}^{n} p_i I(s_i) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Plot of Entropy



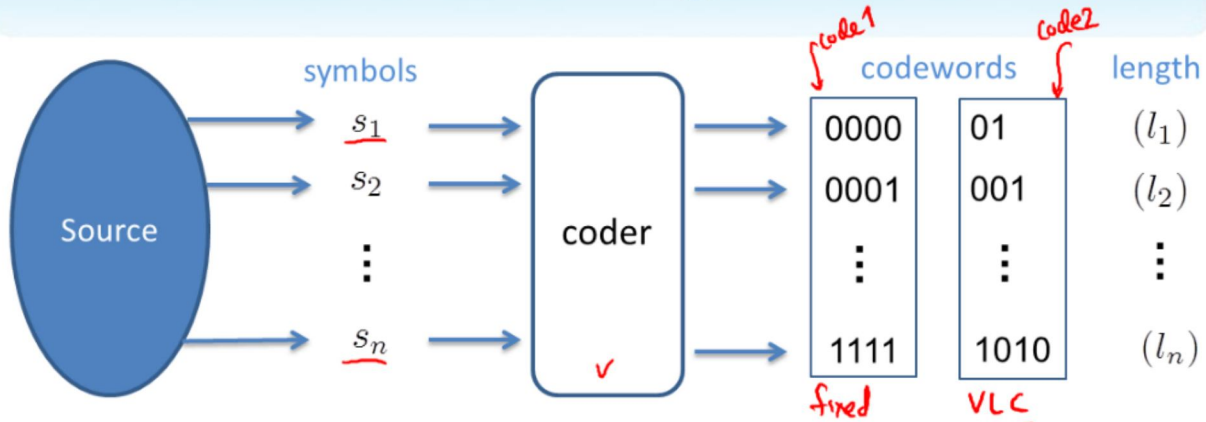
→ $H = -p \log_2 p - (1-p)\log_2(1-p)$

n symbols

H is maximum when $p_i = 1/n$

$H_{max} = \log_2 n$

Coding



Alphabet: $A = \{s_1, s_2, \cdots, s_n\}$

$p_1 \quad p_2 \qquad p_n$

Average Codeword Length: $l_{avg} = \sum_{i=1}^{n} l_i p_i$

Coding

- **First order coders:** encode one symbol at a time, independently of other symbols
- **Block codes:** Group source into blocks of symbols; each block N considered as a single source symbol generated by a source S_N with alphabet size n^N.

  In this case:

$$H(S_N) = N \times H(S)$$

Source Coding Theorem

Let S be a source with alphabet size n and entropy $H(S)$. Consider coding blocks of N source symbols into binary codewords. For any $\delta > 0$, it is possible by choosing N large enough to construct a code in such a way that the average number of bits per original source symbol l_{avg} satisfies

$$H(S) \leq l_{avg} < H(S) + \delta$$

rate

Entropy of a Source

- Entropy of the English language
  - DMS, equal probabilities: H(S)=4.70 bits/letter (4.76 bits/letter for 27 letters)
  - DMS: H(S)=4.14 bits /letter
  - Taking co-dependencies up to 8 letters: H(S)=2.3 bits/letter
- Knowledge about the structure of the data

  Example #1: consider data: 1234323456

  3bits/s : $2^3 = 8$
  - P(2)=P(4)=0.2, P(3)=0.3, P(1)=P(5)=P(6)=0.1 → H=2.44 bits/s
  - Predictor: $x_n = x_{n-1} + r_n$
  - r_n = 1111-1-11111 : P(1)=0.8, P(-1)=0.2 → H=0.7 bits/s
  - *Example #2:* consider data: 334533334545
  - P(3)=1/2, P(4)=P(5)=1/4 →H=1.5 bits/symbol
  - P(33)=P(45)=1/2 → H=1 bit/new symbol →0.5 bits/original symbol

Example

prefix ⇏ UD
prefix ⇒ UD
Instantaneous
UD
Not prefix
Not instantaneous

Symbols	Probability	Code 1	Code 2	Code 3	Code 4
s_1	1/2	0	0	0	0
s_2	1/4	0	1	10	01
s_3	1/8	1	00	110	011
s_4	1/8	10	11	111	0111
Average length		1.125	1.25	1.75	1.875

Entropy H(S)=1.75 bits/symbol

- **Uniquely decodable (UD)** or uniquely decipherable: any finite sequence of codewords corresponds to only one message sequence
- **Prefix**(-free) **code:** no codeword is prefix to another

Binary Tree Representation

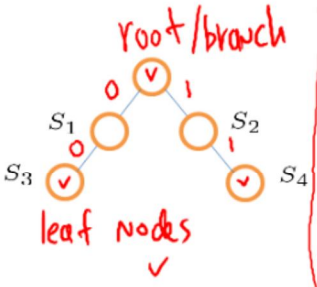
Prefix

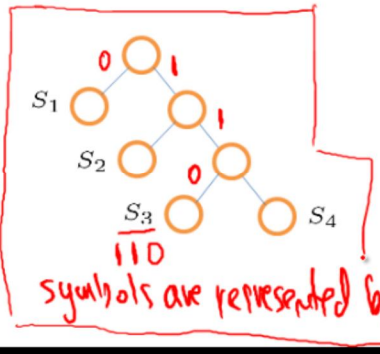
Code 2
0
1
00
11

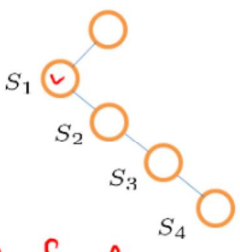
Code 3
0
10
110
111

Code 4
0
01
011
0111

Base of the tree: **root node**
Point at which a branch divides: **branch node**
Termination point: **leaf (or external) node**
In a prefix code, codewords are **only** associated with leaves







Prefix and UD Codes

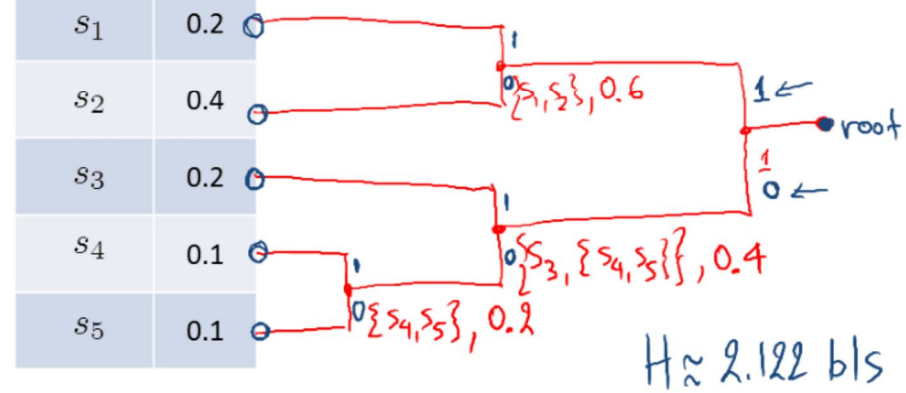
- In general, it is hard to tell if a code is UD (systematic procedure exists to determine if a code is UD in finite number of steps)
- Prefix codes are UD (the converse is not true)
- Prefix codes are instantaneously decodable
- Prefix codes: easy to design, easy to decode

Huffman Coding Algorithm (1951)

- Symbols that occur more often will have shorter codewords Morse principle
- The two symbols that occur least frequently will have the same length
- Design procedure:
  - Sort the symbols according to probability
  - Combine the least probable symbols to form a composite symbol, with probability equal to the sum of the respective symbol probabilities (with this step the size of the alphabet is reduced by one).
  - Repeat procedure for remaining symbols size of alphabet is n → (n-1) stages
  - Extract the codewords from the resulting binary tree representation of the code

Huffman Code Example

Symbol	Prob
s_1	0.2
s_2	0.4
s_3	0.2
s_4	0.1
s_5	0.1



s_1: 11
s_2: 10
s_3: 01
s_4: 001
s_5: 000

Huffman code

$H \simeq 2.122$ bls
$l_{avg} = 2.2$ bls

1

Properties of Huffman Codes

$$H(S) \leq \underbrace{R}_{l_{avg}} < H(S) + 1$$

$$\begin{cases} p_{max} < 0.5 \implies R < H(S) + p_{max} \\ p_{max} \geq 0.5 \implies R < H(S) + p_{max} + 0.086 \end{cases}$$

- Best possible code when $p_i = 2^{-i}$: in such a case: $R = H(S)$
- If alphabet size is large, p_{max} is generally small, and Huffman codes perform quite well
- If alphabet size is small and p_{max} is large, the Huffman codes can be quite inefficient

Huffman Code for Extended Source

- **Block codes:** Group source into blocks of symbols; each block N considered as a single source symbol generated by a source S_N with alphabet size n^N. Then: $H(S_N) = N \cdot H(S)$
- Therefore

$$H(S_N) \le R_N < H(S_N) + 1 \implies N \cdot H(S) \le N \cdot R < N \cdot H(S) + 1$$

$$\implies \boxed{H(S) \le R < H(S) + \frac{1}{N}}$$

- Tradeoff: blocking increases number of possible codewords (e.g., for n=3 and N=8, 3^8=6561 codewords)
  - Storage, computation, delay

Example of Extended Source: Case that Works

Symbols	Probability	Code
s_1	0.8	0
s_2	0.02	11
s_3	0.18	10

Huffman code: entropy = 0.816 bits/symbol; average length = 1.2 bits/symbol; redundancy = 0.384 bits/symbol, or 47% of entropy

Letter	Probability	Code
s_1s_1	0.64	0
s_1s_2	0.016	10101
s_1s_3	0.144	11
s_2s_1	0.016	101000
s_2s_2	0.0004	10100101
s_2s_3	0.0036	1010011
s_3s_1	0.1440	100
s_3s_2	0.0036	10100100
s_3s_3	0.0324	1011

The extended alphabet and corresponding Huffman code; average length = 1.7516 bits/new symbol; or average length = 0.8758 bits/original symbol; redundancy = 0.06 bits/symbol, or 7% of entropy

Example of Extended Source: Case that does not Work

Symbols	Probability	Code
s_1	0.95	0
s_2	0.02	11
s_3	0.03	10

Huffman code: entropy = 0.335 bits/symbol; average length = 1.05 bits/symbol; redundancy = 0.715 bits/symbol, or 213% of entropy

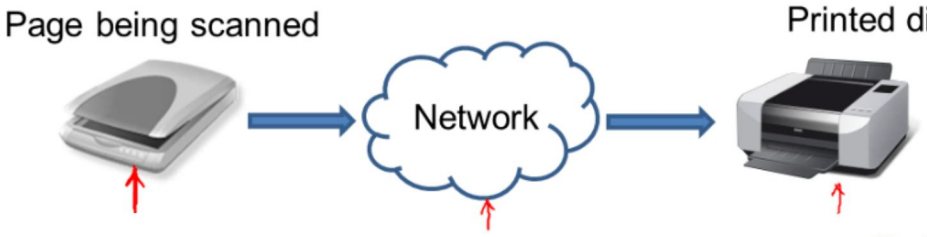
$P_{max} \geq 0.5$ $\quad \bar{l}_{avg} < H(S) + p_{max}$ $\quad 0.086$ $\quad 1.036$

$N = 2$

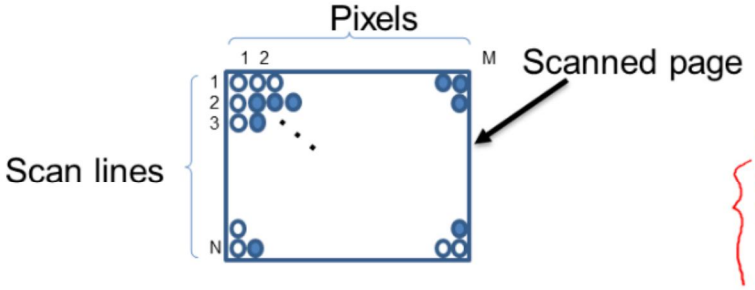
Letter	Probability	Code
$s_1 s_1$	0.9025	0
$s_1 s_2$	0.0190	111
$s_1 s_3$	0.0285	100
$s_2 s_1$	0.0190	1101
$s_2 s_2$	0.0004	110011
$s_2 s_3$	0.0006	110001
$s_3 s_1$	0.0285	101
$s_3 s_2$	0.0006	110010
$s_3 s_3$	0.0009	110000

Huffman code for the extended alphabet; average length = 1.222 bits/new symbol; or average length = 0.611 bits/original symbol; redundancy = 72% of entropy; redundancy drops to acceptable values for N=8 (alphabet size = 6561)

Facsimile Encoding



CCITT (ITU-T) G1 – G4

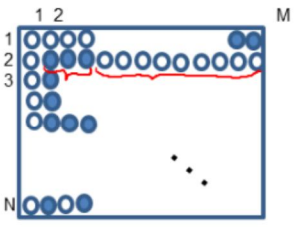


Resolution = M x N
Pixel resolution = 8 per mm
Line resolution = 3.85/7.7 per mm
= 100/200 per inch
A4 document: 210x297 mm

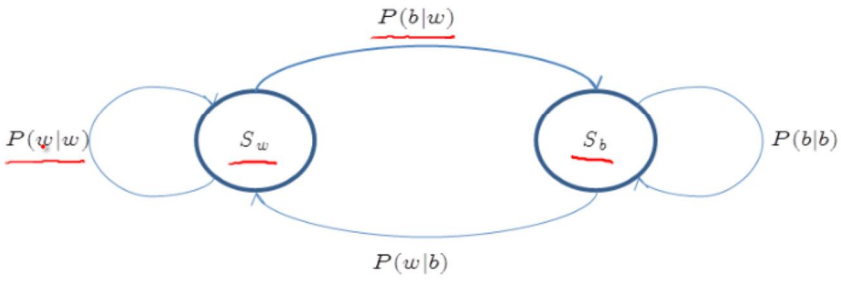
G3, G4 Facsimile Standards

- Developed for text documents
  - Do not work as well for tables, charts, etc.
  - Efficiency drops dramatically for halftone images
- G3: digital operation with analog modems
  - **Run length coding** (1-D and 2-D)
  - **Huffman** compression for black and white runs (static Huffman tables)
  - 2-D: current line predicted from previous line
  - 1-D/2-D mixed to prevent error propagation
- G4: all-digital for digital networks (e.g., ISDN)
  - 2-D run length only (no error protection necessary)

Run Length Coding



A two-state Markov model for binary images



$$H = P(S_w)H(S_w) + P(S_b)H(S_b)$$

G3 Facsimile Standards

- Designing 1-D Huffman codes
  - Estimate probabilities of runs of white pixels and runs of black pixels
  - Follow Huffman encoding procedure
- 1-D scheme (**Modified Huffman** - MH):
  - Must limit the number of possible lengths
  - $r_l = 64m+t$, $t = 0,1,\ldots,63$, $m = 1, 2,\ldots,27$
  - Use codes for m (*make-up codes*) and t (*terminating codes*)
  - Maximum length 1728 (A4-size document)
  - Optional codes for wider documents

ITU-T G3, G4 Fax Huffman Codes

terminating codes →

$r_l = 64 \cdot m + t$

Runlength of 12 white pixels
0010000

runlength of 140 black pixels

$140 = 2 \cdot 64 + 12$
(128)

000011 00 10000000111

White run-length	Code-word	Black run-length	Code-word
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
11	01000	11	0000101
12	001000	12	0000111
13	000011	13	00000100
14	110100	14	00000111
15	110101	15	000011000
16	101010	16	000001011 1
17	101011	17	000001 10000
18	0100111	18	000000 1000
19	0001100	19	000011 00111
20	0001000	20	000011 01000
21	0010111	21	000011 01100
22	0000011	22	000001 10111
23	0000100	23	000001 01000
24	0101000	24	000000 10111
25	0101011	25	000001 11000
26	0010011	26	000001 001010
27	0100100	27	000001 001011
28	0011000	28	000001 001100
29	00000010	29	000001 001101
30	00000011	30	000001 101000
31	00011010	31	000001 101001
32	00011011	32	000001 101010
33	0010010	33	000001 101011
34	00010011	34	000011 010010
35	00010100	35	000011 010011
36	00010101	36	000011 010100
37	00010110	37	000011 010101
38	00010111	38	000011 010110
39	00101000	39	000011 010111
40	00101001	40	000001 101100
41	00101011	41	000001 101101
42	00101011	42	000011 011 1010
43	00101100	43	000011 011 1011
44	00101101	44	000001 010100
45	00000100	45	000001 010101
46	00000101	46	000001 010110
47	00001010	47	000001 010111
48	00001011	48	000001 100100
49	01010010	49	000001 100101
50	01010011	50	000001 010010
51	01010100	51	000001 010011
52	01010101	52	000000 100100
53	00100100	53	000000 110111
54	00101011	54	000000 111000
55	01011000	55	000000 100111

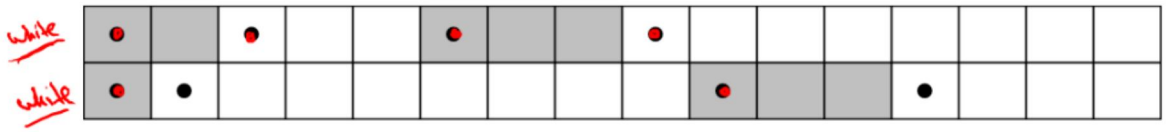
White run-length	Code-word	Black run-length	Code-word
56	01011001	56	000000 101000
57	01011010	57	000000 101100
58	01011011	58	000000 101001
59	01001010	59	000000 101011
60	01001011	60	000000 101 100
61	00110010	61	000001 011010
62	00110011	62	000001 100110
63	00110100	63	000001 100111

make-up codes ←

White run-length	Code-word	Black run-length	Code-word
m=1 → 64	11011	64	0000001111
128	10010	128	000011 001000
192	010111	192	000011 001001
256	0110111	256	000001 101111
320	00110110	320	000001 110011
384	00110111	384	000001 110100
448	01100100	448	000001 110101
512	01100101	512	000000 1101100
576	01101000	576	000000 1101101
640	01100111	640	000000 1001010
704	01101100	704	000000 1001011
768	01100101	768	000000 1001100
832	01101010	832	000000 1001101
896	01101011	896	000000 1110010
960	01101100	960	000000 1110011
1024	01101101	1024	000000 1110100
1088	01101110	1088	000000 1110101
1152	01101111	1152	000000 1110110
1216	01011000	1216	000000 1110111
1280	01011001	1280	000000 1010010
1344	01011010	1344	000000 1010011
1408	01011011	1408	000000 1010100
1472	01001000	1472	000000 1010101
1536	01001001	1536	000000 1011010
1600	01001010	1600	000000 1011011
1664	01101000	1664	000000 1100100
m=27 → 1728	01001011	1728	000000 1100101
1792	00000001000	1792	00000001000
1856	00000001100	1856	00000001100
1920	00000001101	1920	00000001101
1984	00000010010	1984	000000010010
2048	00000010011	2048	00000010011
2112	00000010100	2112	00000010100
2176	00000010101	2176	00000010101
2240	00000010110	2240	00000010110
2304	00000010111	2304	00000010111
2368	00000011100	2368	00000011100
2432	00000011101	2432	00000011101
2496	00000011110	2496	00000011110
m=40 → 2560	00000011111	2560	00000011111
EOL	000000000001	EOL	000000000001

G3, G4 Facsimile Standards

2-D codes



1st row : 0, 2, 3, 3, 8 || 1, 3, 6, 9

2nd row : 0, 1, 8, 3, 4 || 1, 2, 10, 13

- 2-D scheme (**modified READ** - MR)
  - READ: Relative Element Address Designate
  - Exploit heavy correlation of image rows
  - Code transition points with reference to previous line, or distance from previous point is same line

Arithmetic Coding

block coding

- It is more efficient to generate codewords for groups or sequences of symbols rather than generating a separate codeword for each symbol in the sequence
- With Huffman encoding, in finding a codeword for a *particular* sequence of length m, we need codewords for all possible sequences of length m n^m
- A way to assign codewords to *particular* sequences without having to generate codes for all sequences of that length, is offered by **arithmetic coding**
- In arithmetic coding a unique identifier (or **tag**) is generated for the sequence to be encoded, which corresponds to a binary fraction, which becomes the binary code for the sequence

Generating a Tag

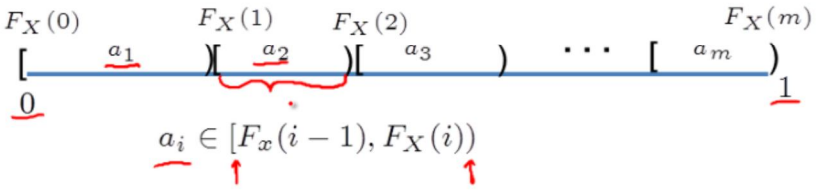
$\mathcal{A} = \{a_1, a_2, \cdots, a_m\}; \quad P(a_i)$ Random variable: $X(a_i) = i$

Probability density function (pdf): $P(X = i) = P(a_i)$

Cumulative distribution function (cdf): $F_X(i) = \sum_{k=1}^{i} P(X = k)$

IDEA: Reduce the size of the interval in which the tag resides as more and more elements of the sequence are received

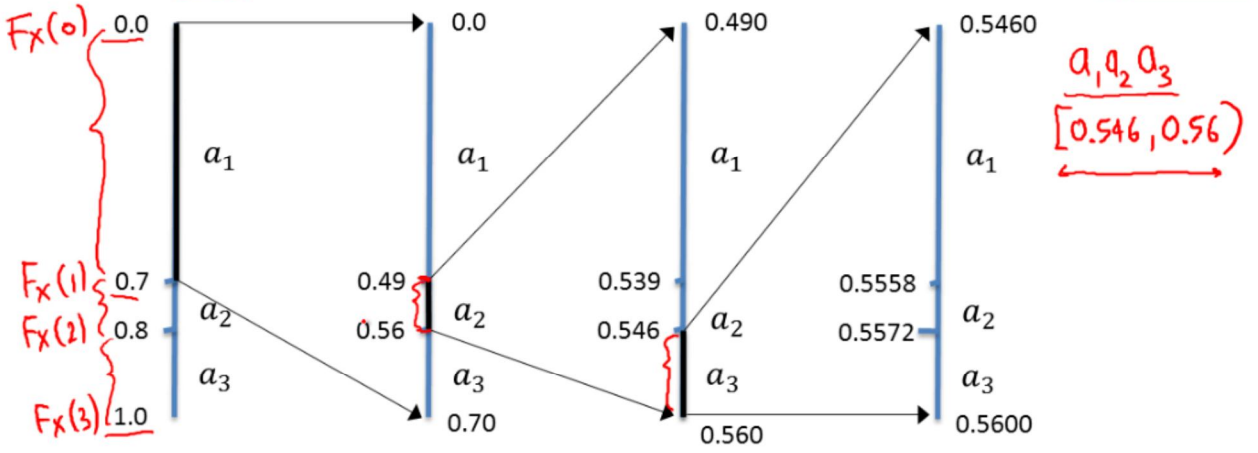
Partition the unit interval into sub-interval of the form



$a_i \in [F_x(i-1), F_X(i))$

Arithmetic Coding Example

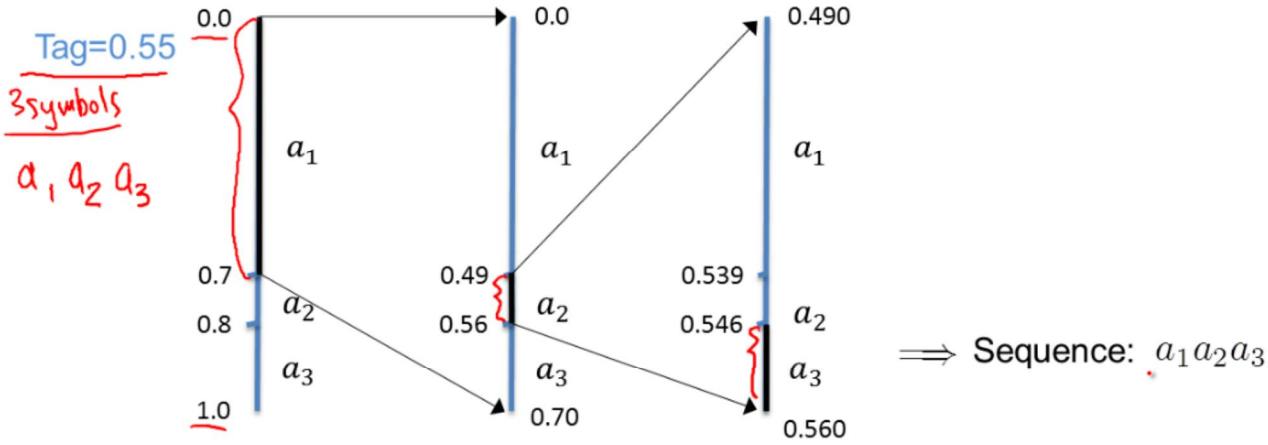
$\mathcal{A} = \{a_1, a_2, a_3\}; \; P(a_1) = 0.7, P(a_2) = 0.1, P(a_3) = 0.2; \; F_X(1) = 0.7, F_X(2) = 0.8, F_X(3) = 1$



K. Sayood, *Introduction to Data Compression*, 2nd edition, Morgan Kaufmann, 2000.

Deciphering the Tag

$\mathcal{A} = \{a_1, a_2, a_3\};\ P(a_1) = 0.7, P(a_2) = 0.1, P(a_3) = 0.2;\ F_X(1) = 0.7, F_X(2) = 0.8, F_X(3) = 1$



Mimic the encoder for decoding

AC vs Huffman

$$H(S) \leq R_{Huffman} < H(S) + \frac{1}{N}$$

$$H(S) \leq R_{AC} < H(S) + \frac{2}{N}$$

Scaling and incremental coding

Taking Context into Account

context - content

Assume a given document has: $P(B) = 0.2, \quad P(W) = 0.8$

$$H = -0.2\log(0.2) - 0.8\log(0.8) = 0.722 \text{ bpp}$$

Divide document into 2 regions:

Region 1 includes 80% of pixels

$P(W) = 0.95, \quad P(B) = 0.05$

$H_1 = 0.286$ bpp

Region 2 includes 20% of pixels

$P(B) = 0.7, \quad P(W) = 0.3$

$H_2 = 0.881$ bpp

Total Entropy: $H = 0.8H_1 + 0.2H_2 = 0.405\text{bpp}$

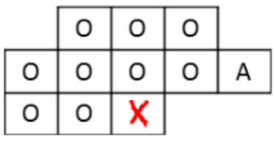
JBIG

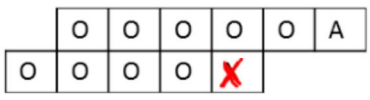
Joint Bi-Level Image Processing Group (JBIG)

Standard for the progressive transmission of bi-level images

$2^{10} = 1024$

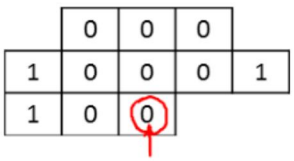
4096

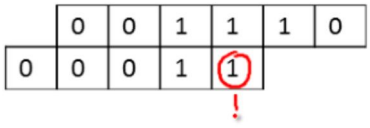




10-pixel neighborhoods

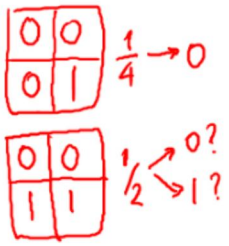
70% B (0)
30% N (1)

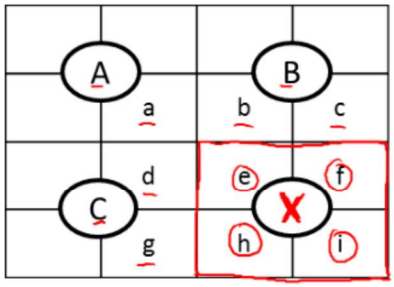




60% B
40% W

JBIG-Progressive Transmission

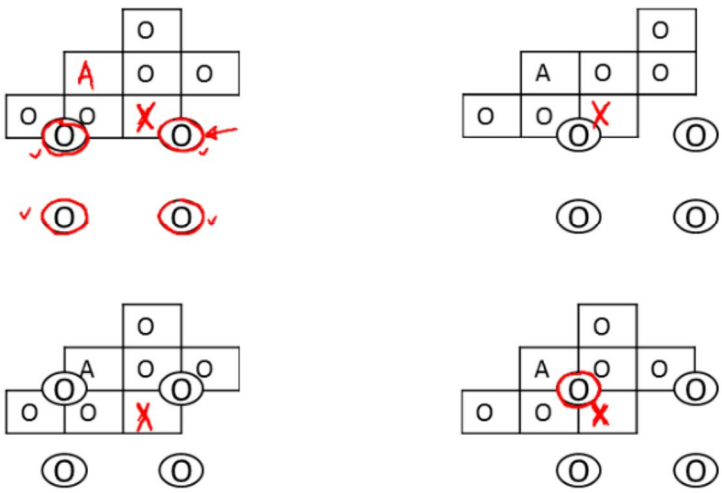




Pixels used for downsampling

$$4e + 2(b + d + f + h) + (a + c + g + i) - 3(B + C) - A \begin{cases} > 4.5 \implies X = 1 \\ < 4.5 \implies X = 0 \end{cases}$$

JBIG-Lossless Compression Revisited



$2^{12} = 4096$

Context templates used in the coding of higher-resolution layers

Dictionary Techniques

- In many applications, the output of the source consists of recurring patterns
- A very reasonable approach to encode such sources is to keep a list, or *dictionary*, of frequently occurring patterns.
- The input is split into two classes, frequently occurring and infrequently occurring patterns
- There are static and adaptive dictionary techniques
- Most adaptive techniques have their roots in two papers by Ziv and Lempel in 1977 (LZ77) and 1978 (LZ78)

Static Dictionary

Code	Entry
000	a
001	b
010	c
011	d
100	r
101	ab
110	ac
111	ad

digrams

$$\mathcal{A} = \{a, b, c, d, r\}$$

a b r a c a d a b r a

101 100 111 101 000

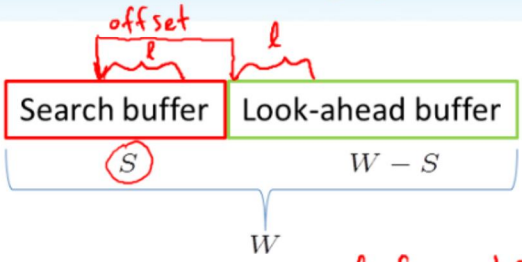
100 100

11 x 3 = 33 bits

7 x 3 = 21 bits

K. Sayood, *Introduction to Data Compression*, 2nd edition, Morgan Kaufmann, 2000.

LZ77 Approach



Encode: $\langle o,\ l,\ c \rangle$

code for next symbol

offset — length of match

Bits needed (using fixed length codes)

ceiling → $\lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 |\mathcal{A}| \rceil$

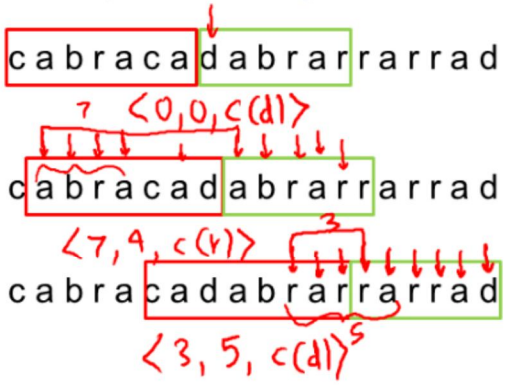
LZ77 Example

S=7 L=6

c a b r a c a d a b r a r r a r r a d

$\langle o,\ l,\ c \rangle$

Example: Encoding



Example: Decoding

c a b r a c a <0,0,c(d)>

c a b r a c a d <7,4,c(r)>

c a b r a c a d a b r a r <3,5,c(d)>

c a b r a c a d a b r a r r a r r a d

LZ78

Drops reliance on search buffer and keeps an EXPLICIT dictionary

This dictionary has to be built at both the encoder and decoder, and therefore they have to be identical

Inputs encoded as: $\langle i, c \rangle$

Code for symbol after matched part

Index in D

$(i = 0 \implies$ New entry in dictionary)

LZ78 Example

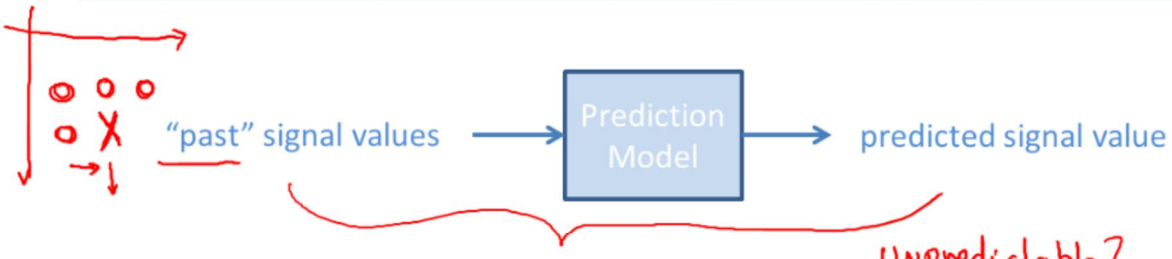
b a w w a / b a w w a / b a w w a / b a w w a / b o o / b o

Encoder output	Index	Entry
<0,c(b)>	1	b
<0,c(a)>	2	a
<0,c(w)>	3	w
<3,c(a)>	4	wa
<0,c(/)>	5	/
<1,c(a)>	6	ba
<3,c(w)>	7	ww
<2,c(/)>	8	a/
<6,c(w)>	9	baw
<4,c(/)>	10	wa/
<9,c(w)>	11	baww
<8,c(b)>	12	a/b
<0,c(o)>	13	o
<13,c(/)>	14	o/
<1,c(o)>	15	bo

Dictionary Techniques

- Variations of LZ77
  - Efficient encoding of the triplets
  - Variable size of S and L buffers
  - Eliminate 3rd element in triplet by adding a flag when a single symbol is encountered
- Variation of LZ78
  - LZW: encoder only sends index
- Applications of LZx
  - Unix *compress*
  - GIF (Graphics Interchange Format)
  - Compression over modems – V.42 bis

Predictive Coding



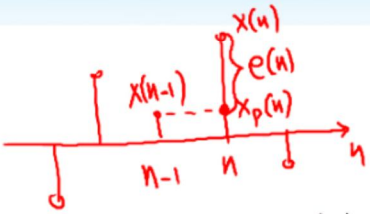
Reconstruction = Prediction + Prediction Error

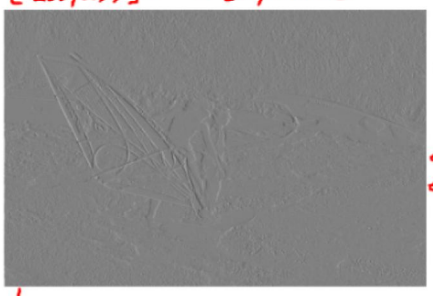
Unpredictable

Context Adaptive Lossless Image Compression (CALIC)

Joint Photographic Experts Group (JPEG)-LS

Prediction Example





predicted value of $x(n)$:

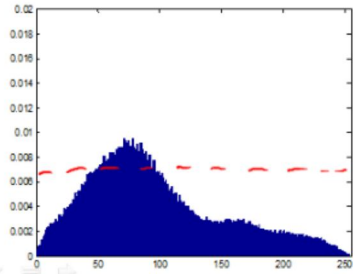
$$x_p(n) = x(n-1)$$

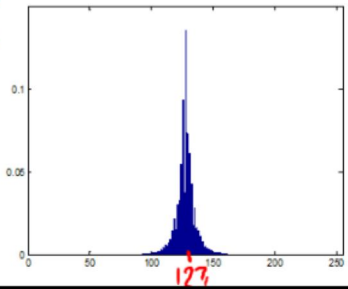
prediction error:

$$e(n) = x(n) - x_p(n)$$

$$= x(n) - x(n-1)$$

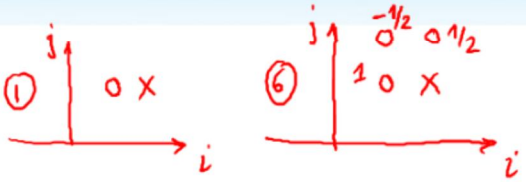
$H_1 > H_2$





JPEG-LS Predictors

Predictors:



0	No prediction
1	$\hat{f}(i,j) = f(i-1,j)$
2	$\hat{f}(i,j) = f(i,j-1)$
3	$\hat{f}(i,j) = f(i-1,j-1)$
4	$\hat{f}(i,j) = f(i,j-1) + f(i-1,j) - f(i-1,j-1)$
5	$\hat{f}(i,j) = f(i,j-1) + (f(i-1,j) - f(i-1,j-1))/2$
6	$\hat{f}(i,j) = f(i-1,j) + (f(i,j-1) - f(i-1,j-1))/2$
7	$\hat{f}(i,j) = (f(i,j-1) + f(i-1,j))/2$

$e(i,j) = f(i,j) - \hat{f}(i,j)$

Correction encoded by adaptive arithmetic coding

JPEG-LS Example

Image	JPEG 0	JPEG 1	JPEG 2	JPEG 3	JPEG 4	JPEG 5	JPEG 6	JPEG 7
Sena	53,431	37,220	31,559	38,261	31,055	**29,742**	33,063	32,179
Sensin	58,306	41,298	37,126	43,445	**32,429**	33,463	35,965	36,428
Earth	38,248	32,295	**32,137**	34,089	33,570	33,057	33,072	32,672
Omaha	56,061	**48,818**	51,283	53,909	53,771	53,520	52,542	52,189

Compressed file size in bytes of the residual images obtained using the various JPEG prediction modes

Image	Best JPEG-LS	GIF
Sena	31,055	51,085
Sensin	32,429	60,649
Earth	32,137	34,276
Omaha	48,818	61,341

Comparison of the file sizes obtained using JPEG lossless compression and GIF

K. Sayood, *Introduction to Data Compression*, 2nd edition, Morgan Kaufmann, 2000.