Data Compression

Why we need data compression?

- Fax machine: 40000 DPSI => 4 million dots per page
- 56 KBPS modem, time to transmit = ?
- Video: 30 pictures per second
- Each picture = 200,000 dots or pixels
- 8-bits to represent each primary color
- Bits required for one picture = ?
- Two hour movie requires = ?

Introduction

- Compression is a way to reduce the number of bits in a frame but retaining its meaning.
- Decreases space, time to transmit, and cost
- Technique is to identify *redundancy* and to eliminate it
- If a file contains only capital letters, we may encode all the 26 alphabets using 5-bit numbers instead of 8-bit ASCII code

Introduction

- If the file had n-characters, then the savings = (8n-5n)/8n => 37.5%

Frequency Dependent Codes

- Not all the characters appear with same frequency, some are more prevalent than the others
- Frequently appearing characters could be assigned shorter codes than the others => results in reduced number of bits
- Such codes are examples of frequency dependent code

Frequency Dependent Codes

- **Huffman code**: (illustrated with a manageable example)

Letter	**Frequency (%)**
A	25
B	15
C	10
D	20
E	30

Frequency Dependent Codes

- **Huffman code**: Code formation
- Assign weights to each character
- Merge two lightest weights into one root node with sum of weights (if multiple? Not unique code)
- Repeat until one tree is left
- Traverse the tree from root to the leaf (for each node, assign 0 to the left, 1 to the right)

Frequency Dependent Codes

- **Huffman code**: Code Interpretation

- No prefix property: code for any character never appears as the prefix of another code (Verify)
- Receiver continues to receive bits until it finds a code and forms the character
- 01110001110110110111 (extract the string)

Frequency Dependent Codes

- **Arithmetic compression**: is based on Interpreting a character-string as a single real number

Letter	Frequency (%)	Subinterval [p, q]
A	25	[0, 0.25]
B	15	[0.25, 0.40]
C	10	[0.40, 0.50]
D	20	?
E	30	?

Frequency Dependent Codes

- **Arithmetic compression**: Coding 'CABAC'
- Generate subintervals of decreasing length, subintervals depend uniquely on the string's characters and their frequencies.
- Interval [x, y] has width w = y – x, the new interval based on [p, q] is x = x + w.p, y = x + w.q
- Step 1: 'C' 0..........0.4.......0.5.............1

based on p = 0.4, q = 0.5

Frequency Dependent Codes

- Step 2: 'A' 0.4.........0.425.......................0.5

based on p = 0.0, q = 0.25

- Step 3: 'B'

  0.4......0.40625.........0.41...........0.425

based on p = 0.25, q = 0.4

Step 4: 'A'

Step 5: 'C'

...0.406625... 0.4067187...

Final representation (midpoint)?

Frequency Dependent Codes

- **Arithmetic compression**: Extracting 'CABAC'

N	Interval[p, q]	Width	Character	N-p	(N-p)/width
0.4067	0.4 – 0.5	0.1	C	0.0067	0.067
0.067	0 – 0.25	0.25	A	0.067	0.268
0.268	0.25 – 0.4	0.15	B	0.018	0.12
?					
?					

When to stop? A terminal character is added to the original character set and encoded. During decompression, once it is encountered the process stops.

Run Length Encoding

- Huffman code requires:

  - frequency values

  - bits are grouped into characters or units

Many items do not fall into such category

  - machine code files

  - facsimile Data (bits corresponding to light or dark area of a page)

  - video signals

Run Length Encoding

- For such files, RLE is used.
- ‘Instead of sending long runs of ‘0’s or ‘1’s, it sends only how many are in the run.’
- 70%-80% space is white on a typed character space, so RLE is useful.

A

Run Length Encoding

- Runs of the same bit
- In facsimile Data, there are many ‘0’s (white spots) –> transmit the run-length as fixed size binary integer
- Receiver generates proper number of bits in the run and inserts the *other* bit in between
- 14 zeros, 1, 9 zeros, 11, 20 zeros, 1, 30 zeros, 11, 11 zeros (number of zeros encoded in 4-bits)

Run Length Encoding

- Runs of the same bit
- Code: 1110 1001 0000 1111 0101 1111 1111 0000 0000 1011
- (next value after 1111 is added to the run)
- SAVINGS IN BITS: ?
- If the stream started with ‘1’ instead?
- Best when there are many long runs of zeros, with increased frequency of ‘1’s, becomes less efficient.

Run Length Encoding

- Runs with different characters
- Send the actual character with the run-length
- HHHHHHHUFFFFFFFFFYYYYYYYYYYYDGGGGG
- code = 7, H, 1, U, 9, F, 11, Y, 1, D, 5, G
- SAVINGS IN BITS (considering ASCII): ?

Run Length Encoding

- Facsimile Compression
- ITU standard (A4 document, 210 by 297 mm)
- 1728 pixels per line
- If 1 bit for each pixel, then over 3 million bits for each page
- A typical page contains many consecutive white or black pixels -> RLE

Run Length Encoding

- Run lengths may vary from 0 to 1728 -> many Possibilities and inefficiency with a fixed size code
- Some runs occur more frequently than others, e.g. most typed pages contain 80% white pixels, spacing between letters is fairly consistent
- => probabilities of certain runs are predictable
- => Frequency dependent code on run lengths

Run Length Encoding

- Some Facsimile compression codes (Terminating, less than 64)

Pixels in the run	Code: White	Code: Black
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
10	00111	0000100
20	0001000	00001101000

Run Length Encoding

- Some Facsimile compression codes (Make up, greater than or equal to 64)

Pixels in the run	Code: White	Code: Black
64	11011	0000001111
128	10010	000011001000
256		
512		

129 white: Savings:

No-prefix property, better compression for long-runs

Relative Encoding

- Relative Encoding:
- Some applications may not benefit from the above: video image -> little repetitive within, but much repetition from one image to the next
- Differential encoding is based on coding only the difference from one to the next

Relative Encoding

- Relative Encoding:

1 2 3 4	1 3 3 4	0 1 0 0
2 5 3 7	2 5 3 7	0 0 0 0
3 6 4 8	3 6 4 7	0 0 0 -1
4 7 5 9	3 7 5 9	-1 0 0 0
1st Frame	2nd Frame	Difference

Resulting difference can be RLE.

Image Representation

- BW pixels each represented by 8-bit level
- Color composed of R, G, B primaries, each is represented by 8-bit level

-> Each color pixel can be represented by one of

$2^8 . 2^8 . 2^8 = 2^{24}$ colors

VGA screen: 640 * 480 pixels

-> 640 * 480 * 24 = 7, 372, 800 bits

Image Compression

- JPEG compression – both for grayscale and color images
- Previous compression methods were lossless – it was possible to recover all the information from the compressed code
- JPEG is lossy: image recovered may not be the same as the original

JPEG Compression

- It consists of three phases: Discrete Cosine Transform (DCT), Quantization, Encoding.
- DCT:

  Image is divided into blocks of 8*8 pixels

  For grey-scale images, pixel is represented by 8-bits

  For color images, pixel is represented by 24-bits or three 8-bit groups

JPEG Compression

- DCT takes an 8*8 matrix and produces another 8*8 matrix.
- $T[i][j] = 0.25\, C(i)\, C(j) \sum \sum P[x][y] \cos (2x+1)i\pi/16 * \cos (2y+1)j\pi/16$

$i = 0, 1, \ldots 7, j = 0, 1, \ldots 7$

$C(i) = 1/\sqrt{2}, i = 0$

$\quad = 1$ otherwise

T contains values called 'Spatial frequencies'

JPEG Compression

- Spatial frequencies directly relate to *how much the pixel values change* as a function of their positions in the block
- T[0][0] is called the DC coefficient, related to average values in the array/matrix, Cos 0 = 1
- Other values of T are called AC coefficients, cosine functions of higher frequencies

JPEG Compression

- Case 1: all P's are same => image of single color with no variation at all, AC coefficients are all zeros.
- Case 2: little variation in P's => many, not all, AC coefficients are zeros.
- Case 3: large variation in P's => a few AC coefficients are zeros.

JPEG Compression

P-matrix	T-matrix
20 30 40 50 60 ...	720 -182 0 -19 0 ...
30 40 50 60 70	-182 0 0 0 0
40 50 60 70 80	0 0 0 0 0
50 60 70 80 90	-19 0 0 0 0
60 70 80 90 100	0 0 0 0 0
...	...

'Uniform color change and little fine detail, easier to compress after DCT'

JPEG Compression

P-matrix	T-matrix
100 150 50 100 100 ...	835 15 -17 59 5...
200 10 110 20 200	46 -60 -36 11 14
10 200 130 30 200	-32 -9 130 105 -37
100 10 90 190 120	59 -3 27 -12 30
10 200 200 120 90	50 -71 -24 -56 -40
....	....

‘Large color change, difficult to compress after DCT ’

JPEG Compression

- To restore, inverse DCT (IDCT) is performed:
- P[x][y] = 0.25 ∑ ∑ C(i) C(j) T[i][j] Cos (2x+1)iπ/16 * Cos (2y+1)jπ/16

x = 0, 1, …7, y = 0, 1, …7

\# Can write a C-program to apply DCT on a P-array (8*8) to obtain T-array and also IDCT on T-array to recover P-array.

JPEG Compression

- Quantization: Provides an way of ignoring small differences in an image that may not be perceptible.
- Another array Q is obtained by dividing each element of T by some number and rounding-off to nearest integer => loss

JPEG Compression

T-matrix	Q-matrix
152 0 -48 0 -8...	15 0 -5 0 -1...
0 0 0 0 0	0 0 0 0 0
-48 0 38 0 -3	-5 0 4 0 0
0 0 0 0 0	0 0 0 0 0
-8 0 -3 0 13	-1 0 0 0 1
...	...

‘Divide by 10 and round-off’ => ‘creates fewer distinct numbers and more consistent pattern’

JPEG Compression

- Can we recover by multiplying the elements with 10?
- If the loss is minimal, the vision system may not notice.
- Dividing T-elements by the same number is not practical, may result in too much loss.
- Retain the effects of lower spatial frequencies as much as possible – less subtle features noticed if changed

JPEG Compression

- Upper left elements to be divided with smaller values
- Lower right elements – higher spatial frequencies, finer details - to be divided with larger values
- To define a quantization array U, then

Q[i][j] = Round (T[i][j] ÷ U[i][j]), i = 0, 1, …7, j = 0, 1, …7

JPEG Compression

- U = 1 3 5 7 9… Q= 152 0 -10 0 -1 …

U					Q				
1	3	5	7	9…	152	0	-10	0	-1 …
3	5	7	9	11	0	0	0	0	0
5	7	9	11	13	-10	0	4	0	0
7	9	11	13	15	0	0	0	0	0
9	11	13	15	17	-1	0	0	0	0
…					…				

‘Q can be well-compressed due to redundant elements’

JPEG Compression

- Encoding
  - linearize two dimensional data and compress it

152 0 -10 0 -1 ...

0	0	0	0	0
-10	0	4	0	0
0	0	0	0	0
-1	0	0	0	0
...				

- Row-wise (shorter runs)
- Zigzag (longer runs, higher spatial frequencies are gathered together)

JPEG Compression

- Many such 8*8 arrays, adjacent blocks with little difference => more potential for compression
- JPEG may provide 95% compression (depends on image and quantization array)
- GIF (Graphic Image Format) reduces color to 256. Best suited for a few colors and sharp boundaries (charts, lines). Not good for variations and shading – full color photo

Multimedia Compression

- JPEG compresses still pictures, motion pictures are compressed by MPEG.
- Still picture contains 7, 372,800 bits.
- Compression ratio 20:1 reduces the bits to 368,640
- With 30 images per second, bits to be handled 11,059,200.

=> Huge data transmission if channel is shared

Multimedia Compression

- Based on JPEG compression and relative encoding
- Usually little difference from one to the next frame => suitable for compression
- A completely new scene can not be compressed this way. Also, not suited for moving objects unleashing other objects.
- Three frames: I, P, B.

Multimedia Compression

- I (intra-picture) - frame: Just a JPEG encoded image.
- P (predicted) – frame: Encoded by computing the differences between a current and a previous frame.
- B (bidirectional) - frame: Similar to P-frame except that it is interpolated between previous and future frame.

Multimedia Compression

- I-frames must appear periodically in any frame sequence, otherwise:

  - any error introduced in any frame is propagated in all subsequent frames if relative differences are sent

  - in broadcast applications, if one tunes-in late, then he has nothing to compare to

Multimedia Compression

- Typical sequence:

order in which to transmit: I – P – I

P is sandwiched between groups of B

P is the difference from the previous I-frame

B is interpolated from the nearest I and P

order in which to display I – B – B – P – B – B – I

Multimedia Compression

- Coding P:
- Motion compensated Prediction for P-frames
- Divide the image into macro-blocks of 256 pixels (16*16)
- Chrominance arrays are reduced to groups of 4 pixels (becomes 8*8 matrix) representing average values => loss, not perceptible

Multimedia Compression

- An algorithm examines each macro-block of P-frame and locates a best matching macro-block in the prior I-frame
- Not necessarily in the same relative position (in the vicinity)
- Once located, the algorithm calculates the difference and motion vector (displacement)
- The above information is encoded and transmitted

Multimedia Compression

- At the decoding end, the motion-vector is used to determine the position and the *difference* is used to reconstruct the macro-block

Multimedia Compression

- MP3 (MPEG Level 3) – compression protocol for audio
- Audible range: 20 – 20 KHz
- PCM uses 16 bit samples @ 44.1 KHz
- 1-sec of PCM audio needs: 16*44.1*1000 bits
- For two channel stereo: Twice
- For 1-minute audio: 60 times -> 88 Mbits

Multimedia Compression

- MP3 is based on a ‘psychoacoustic model’ – what we can hear and what we can distinguish
- ‘Auditory masking’ – capture an audio signal, determine what we can not hear, remove those components from the stream, digitize what is left
- ‘Sub-band coding’ – decompose the original signal into non-overlapping frequency ranges, create bit-stream for each band

Multimedia Compression

- Sub-bands are encoded differently:
- Sub-bands with loud signals need good resolution, are coded with more bits
- Near-by sub-bands with weak signals are effectively masked by louder signals, need less resolution, are coded with fewer bits, that results in *compression*