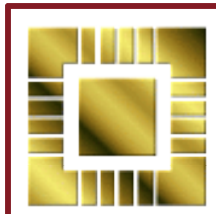
Electronics and Computer Science
University of Southampton

CM214-COMP2008
Data Communications and Networks

Lossless Data Compression

Karl R. Wilcox
krw@ecs.soton.ac.uk



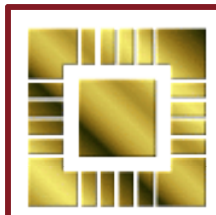
Objectives

- To understand data compression
  - How it works
  - Its role in networks and communications
  - When to use it
  - When not to use it

- (Peterson & Davie, Section 7.2)

Compression

- COMPRESSION
  - Process of encoding data so it is smaller (in time and/or space) than original data.
- WHY COMPRESS?
  - To reduce bandwidth consumption (time / space / cost) on a network
  - To reduce the long-term storage space (archiving)
- Requirements may have different characteristics:
  - speed of encoding important for networks
  - compression ratio more important for archiving.



Lossless Vs Lossy

- Some encodings preserve all the original data
  - "lossless"
  - e.g. Huffman
- others may discard (hopefully insignificant) information
  - "lossy"
  - e.g. JPEG

Why Not Compress? - A

1. If the “cost” of compression does not exceed the benefits, e.g.
   - Small items do not compress well (fixed overheads of compression)
     - Examples: TELNET, SSH
   - No net gain in network transmission time (See next slide)

Cost Vs Benefit

A network with bandwidth B_n bits/sec can transmit x bits in x/B_n

If data (de)compressed at rate of B_c bits/sec with compression ratio r:1 then time to transmit is total of:

Time to compress x bits x/B_c

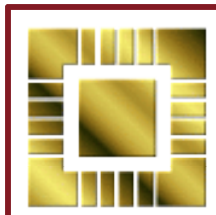
Time to transmit bits $x/(rB_n)$

Time to decompress x/B_c

To be worthwhile total must be less than time to transmit uncompressed data

$$2x/B_c + x/(rB_n) < x/B_n$$

So compression rate B_c must be $> (2B_n) / (1 - 1/r)$



Why Not Compress? – B

1. Already compressed data
   - Cannot be compressed again by same method
2. Random data cannot be compressed
   - i.e. equal probability of occurrence
   - Compression takes advantage of redundancy / duplication in data
   - “uncompressible” is one definition of “random”!

Why Not Compress? – C

1. Compressed data less tolerant of errors
   - Single bit error corrupts entire zip archive
   - Compare to single bit error in ASCII text
2. Lossy compression may lose important data
   - E.g. watermarks in images
   - Steganographic data

Why Not Compress? – D

1. May be more susceptible to attack
   - Recent case of malicious zip file e-mail attachment
   - Mail server virus scanner would uncompress zip file to find it contains a 1Mb zip file
     - This zip uncompressed, to find it contains a 1Mb zip file
       - This zip uncompressed, to find it contains a 1Mb zip file……

Huffman Encoding

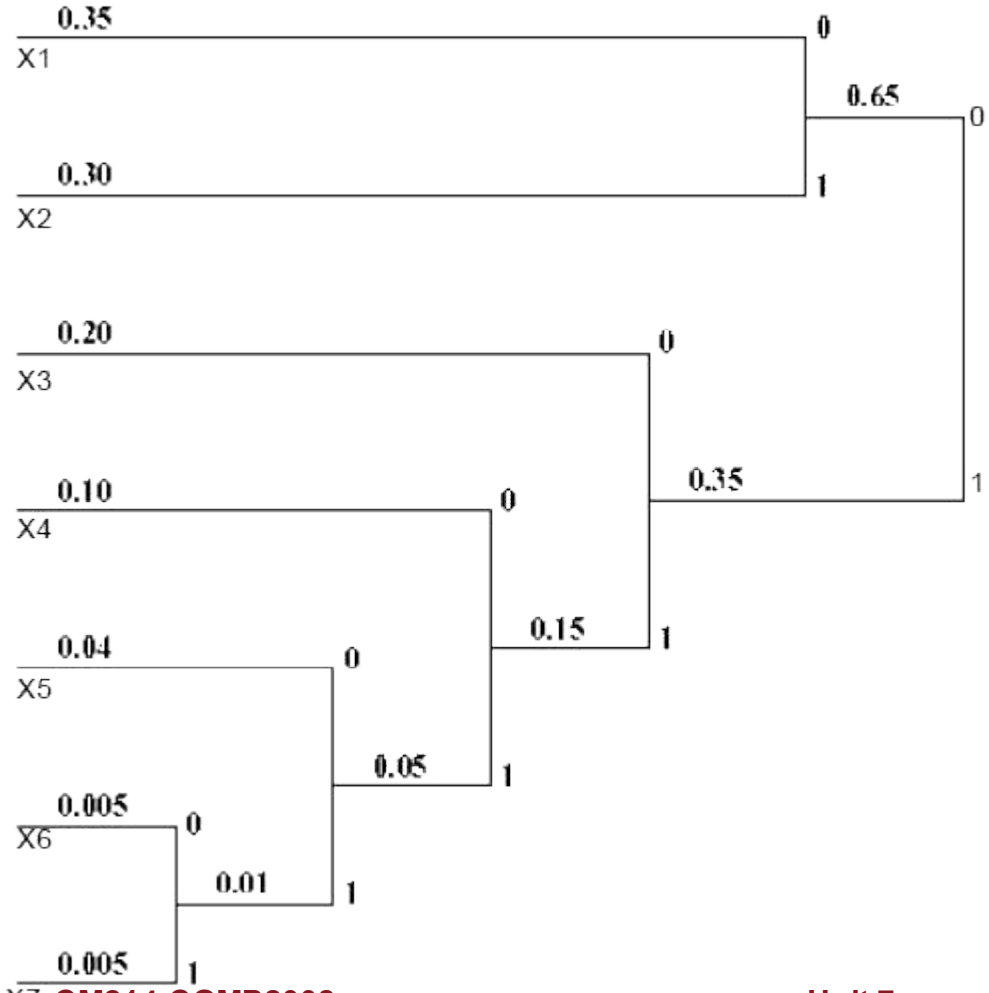
- "Optimal for discrete memoryless sources"
  - i.e. good for human texts!
- Relies on "symbols" (letters) having *different* probabilities of occurrence
  - Constructs binary tree
  - High probability near top of tree (few bits)
  - Low probability near bottom (more bits)

Huffman Algorithm

1. Order symbols into decreasing probability of occurrence
   - $\{X_1, X_2, \ldots X_n\}$ probabilities $\{P_1, P_2, \ldots P_n\}$
2. Combine last two elements to single element with prob. $P'_{n-1} = P_{n-1} + P_n$
3. Append 0 & 1 to last digits of code words for X_{n-1} & X_n

Easier to understand on diagram!

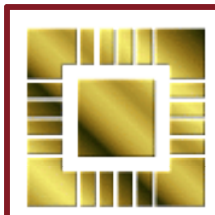
Huffman Example



Sym.	Prob.	Enc.
X_1	0.35	00
X_2	0.30	01
X_3	0.20	10
X_4	0.15	110
X_5	0.04	1110
X_6	0.005	11110
X_7	0.005	11111

Huffman Considerations

- Loses the byte boundary
  - Data becomes a pure bit stream
  - Need to mark end of data (cannot 0 pad)
- Static dictionary
  - E.g. English letter frequencies
  - Do not need to send dictionary
- Dynamic dictionary
  - Calculation & sending involves overhead



Lempel-Ziv Encoding

- Dictionary based, but works on arbitrary bit streams
  - Efficiency increases with longer bitstreams
  - Can rebuild dictionary if it becomes inefficient
  - Used in GIF, ZIP & many others
  - Loses byte boundary again

Lempel-Ziv Encoding

1. Start with an empty dictionary
2. Match the input stream with phrases in the dictionary
3. Create new phrase from old + different end symbol
4. Add phrase to dictionary
5. Encoded output is dictionary position + new letter

Lempel-Ziv Example

- Input stream shown below
  - Commas indicate phrase boundaries
  - Not part of input

1,0,10,11,01,00,100,
111,010,1000,011,
001,110,101,10001,
1011

	Dictionary Location	Dictionary Contents	Code Word
0	0000		
1	0001	1	00001
2	0010	0	00000
3	0011	10	00010
4	0100	11	00011
5	0101	01	00101
6	0110	00	00100
7	0111	100	00110
8	1000	111	01001
9	1001	010	01010
10	1010	1000	01110
11	1011	011	01011
12	1100	001	01101
13	1101	110	01000
14	1110	101	00111
15	1111	10001	10101
16		1011	11101

Lemep-Ziv Features

- Do not need to send dictionary
  - Can be built from input stream
  - (For a given size of dictionary)
  - Can rebuild dictionary if performance falls
  - But need “marker” and “bit stuffing”
- Example actually makes “compressed” version longer
  - On longer bitstreams very efficient

Other Techniques

- Run Length Encoding
  - How many ‘1’s, how many ‘0’s
  - Used in Fax transmission
- Delta Encoding
  - Difference between current “word” & previous
- Many others + variants of above

Compression Comparisons

- No single answer for lossless compression
  - Depends on application & data
- Comparisons (on Unix systems)
  - **`compress`** (Lempel-Ziv)
  - **`pack`** (Huffman, single dictionary)
  - **`compact`** (adaptive Huffman)

Summary

- Lossless compression can reduce network bandwidth usage
  - Sometimes used at packet level in networking (e.g. PPP over slow links)
  - But there is a processing overhead
  - Which may make compression impractical

- Compression is not always appropriate!