Lossless/Near-lossless Compression of Still and Moving Images

Xiaolin Wu
Polytechnic University
Brooklyn, NY

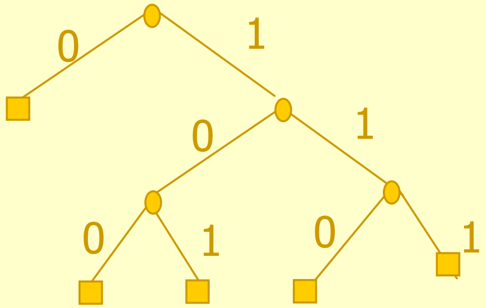
Part 2. Entropy coding

Variable length codes (VLC)

- Map more frequently occurring symbols to shorter codewords
  - abracadabra
    - fixed length a - 000 b - 001 c - 010 d - 011 r - 100
    - variable length a - 0 b - 10 c - 110 d - 1110 r - 1111
- For instantaneous and unique decodability we need prefix condition, i.e. no codeword is prefix of another
  - Non-prefix code 0 01 011 0111
  - Prefix code 0 10 110 111

Optimality of prefix codes

- Optimal data compression achievable by any VLC can always be achieved by a prefix code
- A prefix code can be represented by a labeled binary tree as follows



Prefix code

{0, 100, 101, 110, 111}

- An optimal prefix code is always represented by a full binary tree

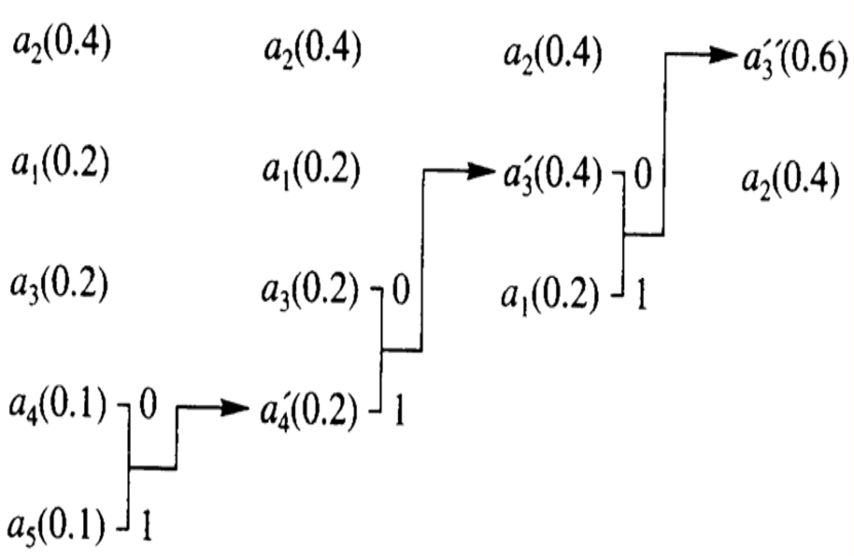
Huffman codes

- Developed in 1952 by D.A. Huffman.
- Let source alphabet be $s_1, s_2, \ldots, s_N$ with probability of occurrence $p_1, p_2, \ldots, p_N$
  - Step 1 Sort symbols in decreasing order or probability
  - Step 2 Merge two symbols with lowest probabilities, say, s_{N-1} and s_N. Replace (s_{N-1}, s_N) pair by H_{N-1} (the probability is $p_{N-1} + p_N$). Now new set of symbols has N-1 members $s_1, s_2, \ldots, H_{N-1}$.
  - Step 3 Repeat Step 2 until all symbols merged.

Huffman codes (contd.)

➔ Process viewed as construction of a binary tree. On completion, all symbols s_i will be leaf nodes. Codeword for s_i obtained by traversing tree from root to the leaf node corresponding to s_i.

Letter	Probability	Codeword
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011



Average code length 2.2

Properties of Huffman codes

- Optimum code for a given data set requires two passes.
- Code construction complexity O(N logN).
- Fast lookup table based implementation.
- Requires at least one bit per symbol.
- Average codeword length is within one bit of zero-order entropy (Tighter bounds are known).
- Susceptible to bit errors.

Huffman codes - Blocking symbols to improve efficiency

- p(w) = 0.8, p(b) = 0.2 Entropy = 0.72

  Bit-rate = 1.0 Efficiency = 72%
- p(ww) = 0.64, p(wb)=p(bw)=0.16, p(bb) = 0.04

  Bit-rate = 0.80 Efficiency = 90%
- Blocking three symbols we get alphabet of size 8 and average bit-rate 0.75 efficiency 95%
- Problem - alphabet size and consequently Huffman table size grows exponentially with number of symbols blocked.

Run-length codes

- Encode runs of symbols rather than symbols themselves

  bbbaaaddddcfffffffaaaaadddd

  encoded as 3b3a4d1c7f5a5d

- Especially suitable for binary alphabet

  001111111000000011011111100000

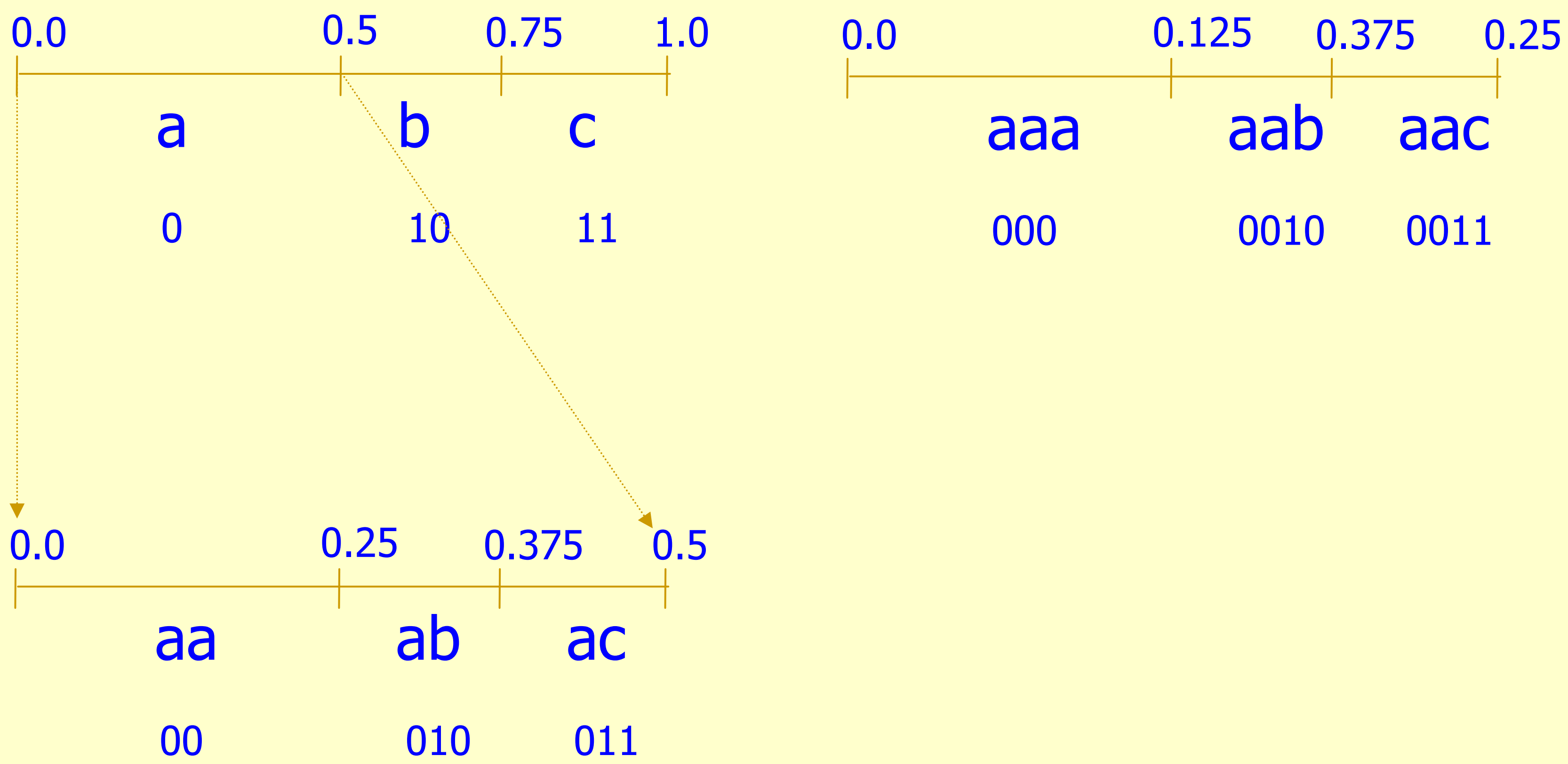
  encoded as 2,7,7,2,1,6,5

- Run lengths can be further encoded using a VLC

Arithmetic Coding

- We have seen that alphabet extension i.e. blocking symbols prior to coding can lead to coding efficiency
- How about treating entire sequence as one symbol!
- Not practical with Huffman coding
- Arithmetic coding allows you to do precisely this
- Basic idea - map data sequences to sub-intervals in (0,1) with lengths equal to probability of corresponding sequence.
- To encode a given sequence transmit any number within the sub-interval

Arithmetic coding - mapping sequences to sub-intervals

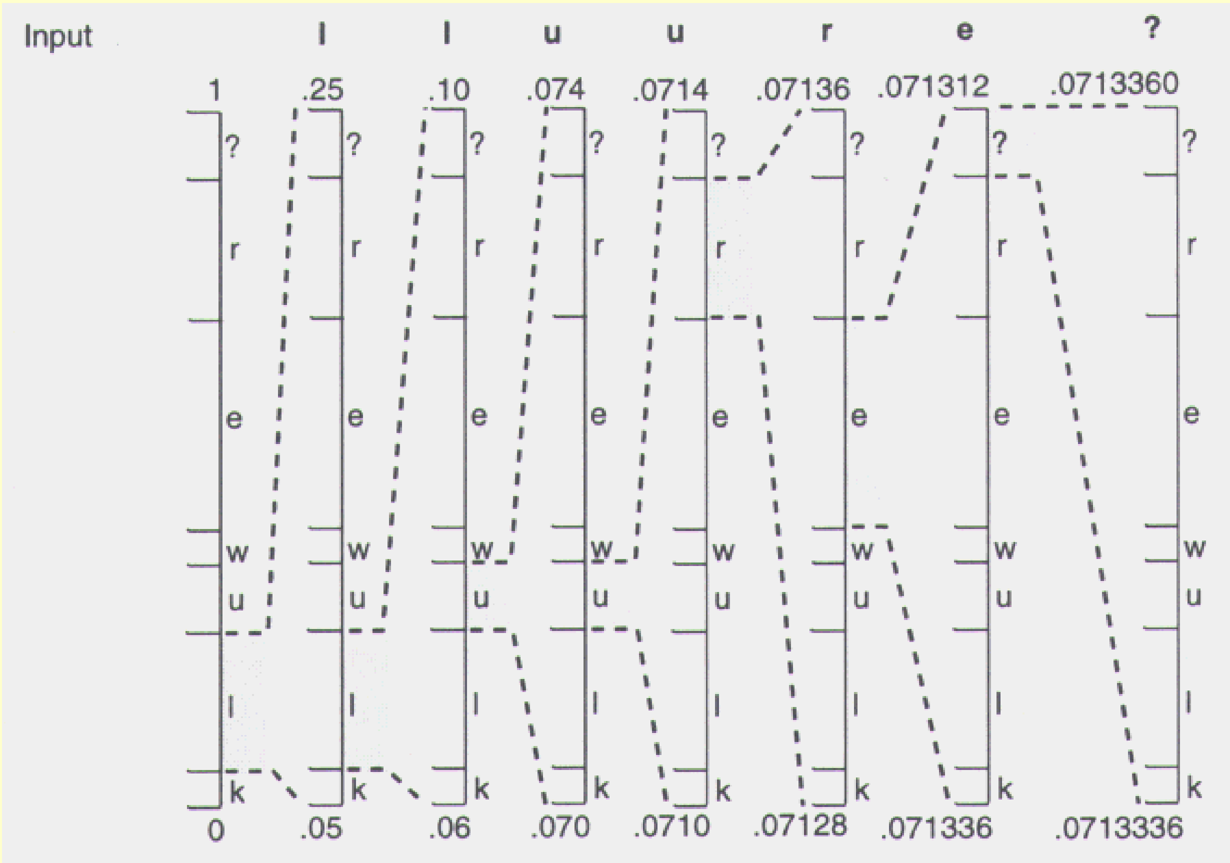


Arithmetic coding - encoding example

Message is lluure?
(we use ? As message terminator)

s_i	p_i	Subinterval
k	0.05	[0.00,0.05)
l	0.20	[0.05,0.25)
u	0.10	[0.25,0.35)
w	0.05	[0.35,0.40)
e	0.30	[0.40,0.70)
r	0.20	[0.70,0.90)
?	0.10	[0.90,1.00)

Initial partition of (0,1) interval



Final range is [0.0713336, 0.0713360). Transmit any number within range, e.g. 0.0713348389… **16 bits**. (Huffman coder needs 18bits. Fixed coder: 21bits).

Arithmetic coding - decoding example

Symbol probabilities

s_i	p_i	Subinterval
k	0.05	[0.00,0.05)
l	0.20	[0.05,0.25)
u	0.10	[0.25,0.35)
w	0.05	[0.35,0.40)
e	0.30	[0.40,0.70)
r	0.20	[0.70,0.90)
?	0.10	[0.90,1.00)

Modified decoder table

s_i	i	$cumprob_i$
k	7	0.00
l	6	0.05
u	5	0.25
w	4	0.35
e	3	0.40
r	2	0.70
?	1	0.90
	0	1.00

lo = 0, hi = 1, range = 1.

1. We find i = 6 such that $cumprob_6$ <= (value-lo)/range < $cumprob_5$
   Thus first decoded symbol is l.
2. Update: hi = 0.25, lo = 0.05, range = 0.2
3. To decode next symbol we find i = 6 such that $cumprob_6$ <= (value - 0.05)/0.2 < $cumprob_5$ thus next decoded symbol is l.
5. Update hi = 0.10, lo = 0.06, range = 0.04.
6. Repeat above steps till decoded symbol is ? Terminate decoding.

Arithmetic coding - implementation issues

- **Incremental output at encoder and decoder**
  - From example discussed earlier, note that after encoding u, subinterval range [0.07, 0.074). So, can output 07.
  - After encoding next symbol, range is [0.071, 0.0714). So can output 1.
- Precision - intervals can get arbitrarily small
  - Scaling - Scale interval every time you transmit
  - Actually scale interval every time it gets below half original size - (this gives rise to some subtle problems which can be taken care of)

Golomb-Rice codes

- Golomb code of parameter *m* for positive integer *n* is given by coding *n div m* (quotient) in binary and *n mod m* (remainder) in unary.
- When *m* is power of *2*, a simple realization also known as Rice code.
- Example: *n = 22, k = 2* (*m = 4*).
  - *n = 22 = '10110'*. Shift right *n* by *k (= 2)* bits. We get *'101'*.
  - Output *5* (for *'101'*) *'0's* followed by *'1'*. Then also output the last *k* bits of *N*.
  - So, Golomb-Rice code for *22* is '*00000110*'.
- Decoding is simple: count up to first *1*. This gives us the number *5*. Then read the next *k (=2)* bits - *'10'*, and *n = m* x *5 + 2 (for '10') = 20 + 2 = 22*.

Comparison

- In practice, for images, arithmetic coding gives 10-20% improvement in compression ratios over a simple Huffman coder. The complexity of arithmetic coding is however 50 - 300% higher.
- Golomb-Rice codes if used efficiently have been demonstrated to give performance within 5 to 10% of arithmetic coding. They are potentially simpler than Huffman codes.
- Multiplication free binary arithmetic coders (Q, QM coders) give performance within 2 to 4% of M-ary arithmetic codes.

Further Reading for Entropy Coding

- Text Compression - T.Bell, J. Cleary and I. Witten. Prentice Hall. Good coverage of arithmetic coding
- The Data Compression Book - M. Nelson and J-L Gailly. M&T Books. Includes source code.
- Image and Video Compression Standards - V. Bhaskaran and K. Konstantinides. Kluwer International. Hardware Implementations.