Principle of Lossless Compression

Dr. C.-C. Jay Kuo
Integrated Media Systems Center
University of Southern California
Los Angeles, CA 90080
E-mail: cckuo@sipi.usc.edu

Contents

- Background
- Huffman coding
- Arithmetic coding
- QM coder
- Lempel Ziv coding
Background (1)

- Basic concept from information theory
  - For symbol $S_k$ with probability $P(S_k)$:
    - information: $I(S_k) = -\log_2 P(S_k)$
    - entropy: average of information:
      $$H = -\sum_{k} P(S_k) \log_2 P(S_k)$$
  - Shannon’s noiseless coding theorem:
    - For a source with entropy $H$ bits/symbol, it is possible to find a distortionless coding scheme using an average of $(H+e)$ bits/message ($e>0$)

Background (2)

- Modeling
  - physical model
  - statistical model
    - probability model
      - statistical model
      - adaptive model
    - Markov state model
    - dictionary model
  - composite model
Variable Length Coding

- Principle
  - to encode symbols with a higher probability (less information) with fewer bits and encode symbols with a lower probability (more information) with more bits
- Its performance is better than fixed length codes
- Well known example: Huffman code

Contents

- Background
- Huffman coding
- Arithmetic coding
- QM coder
- Lempel Ziv coding
Shannon-Fano Code (1)

- Procedures:
  - Sort symbols according to their probabilities
  - Repeatedly divide them into two groups with almost equal probabilities until there is only one symbol left in each group
  - Build the coding tree from the root to leaves according to the sequence of separated lines
  - Set ‘0’ or ‘1’ to each branch

Shannon-Fano Code (2)

<table>
<thead>
<tr>
<th>symbol</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>0.39</td>
<td>0.18</td>
<td>0.15</td>
<td>0.15</td>
<td>0.07</td>
<td>0.06</td>
</tr>
</tbody>
</table>

- 1st
- 2nd
- 3rd
- 4th
- 5th

- root
- separated line
- tree line
Shannon-Fano Code (3)

- Code table:
  - A 00
  - B 01
  - C 10
  - D 110
  - E 1110
  - F 1111

- Entropy:
  \[ H = -\sum_{k=1}^{6} P(r_k) \log_2 P(r_k) = 2.3083 \]

- Bits average:
  \[ B_{\text{avg}} = \sum_{k=1}^{6} P(r_k) L(r_k) = 2.41 \]

- Coding redundancy:
  \[ CR = B_{\text{avg}} - H = 0.1017 \]

Huffman Coding (1)

- Background:
  - Developed by David Huffman as part of a class assignment. The class was taught by Robert Fano at MIT (1951).
  - Huffman coder is a prefix-free coder.
    - No codeword is other codewords’ prefix. (no decoding conflict)
  - Huffman coder is an optimal coder for a given probability model.
Huffman Coding (2)

- Based on the two observations on an optimal prefix coder:
  - symbol that occur more frequently will have shorter codeword than symbol that occur less frequently.
  - the two symbols that occur least frequently will have the same codeword length.
  - because Huffman coder is a prefix coder, it is not necessary to let the codeword of least prob symbol longer than that of second least prob symbol.

Huffman Coding (3)

- Recursive Procedures:
  - sort each symbol by its probability and add the node for each symbol into the coding list.
  - combine the two nodes with least prob to one new node which embedded the sum of these two prob.
  - assign ‘0’ and ‘1’ arbitrarily to the branches of these two nodes with least prob.
  - drop these two nodes from the coding list.
Huffman Coding (4)

- Unlike Shannon-Fano coder which is encoded from root to leaf, Huffman coder encodes from leaf to root.
- Disadvantages:
  - large computation complexity for a large amount of symbols case.
  - static (fixed) Huffman codeword table can severely affect the coding performance.

Huffman Coding (5)

<table>
<thead>
<tr>
<th>symbol:</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob:</td>
<td>0.39</td>
<td>0.18</td>
<td>0.15</td>
<td>0.15</td>
<td>0.07</td>
<td>0.06</td>
</tr>
</tbody>
</table>

```
  0
  / \
 /    \ 0.33
\  0.15 /  \ 1
   \ 0.15 /   0
       \ 0.13 / 1
            \ 0.28 / 1
                \ 0.61 / 1
```

tree line
Huffman Coding (6)

- Code table:
  
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>100</td>
</tr>
<tr>
<td>C</td>
<td>101</td>
</tr>
<tr>
<td>D</td>
<td>110</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>

- entropy:
  \[ H = - \sum_{k=1}^{6} P(r_k) \log P(r_k) = 2.3083 \]

- bits average:
  \[ B_{avg} = \sum_{k=1}^{6} P(r_k) L(r_k) = 2.35 \]

- coding redundancy:
  \[ CR = B_{avg} - H = 0.0417 \]

Modified Algorithm

- Disadvantages in Huffman coding:
  - large computation and sorting in the tree for large amount of symbols.
  - performs poor for large amount of data if using the fixed probability table.

- Modified method:
  - forget factor method (Huffman, Arithmetic).
  - higher order model (Huffman, Arithmetic).
  - adaptive tree (Huffman).
Forget Factor Method (1)

- To separate the data to many different groups and update the estimated probability table by previous group’s statistical results.
- To use the forget factor to eliminate the high frequency effect between different groups.
- Encoder and decoder compute the estimated probability table synchronously.
- Can be used for any probability model based method.

Forget Factor Method (2)

- Procedures:
  - initial setting the estimated probability table by uniform distribution model.
  - to code $K^{th}$ group by the statistical result of the $(K-1)^{th}$ group’s probability plus $(K-2)^{th}$ group’s probability multiplexed by the forget factor, $f$

$$\Pr_j(K) = \frac{\Pr_j(K-1) + f \times \Pr_j(K-2)}{\sum_{i=1}^{L} \Pr_i(K-1) + f \times \Pr_i(K-2)}$$
Higher Order Model:

- Previous probability table based on the order-0 model, i.e. to estimate each symbol’s probability independently.
- Order-N model means to compute the probability of \( \Pr(S_k|S_{t_0}, S_{t_1}, \ldots, S_{t_N}) \).
- Also named as “context model”.
- Example: to compress the text in one book: \( \Pr('e')=0.1732 \) but \( \Pr('e'|'t','h')=0.8344. \)

Adaptive Huffman Coding (1)

- A fixed estimated probability table performs poor for a large amount of data. However, adaptive estimated probability model can achieve better results.
- Based on the causal data information, and having no knowledge about the future statistics, adaptive Huffman coder let the Huffman tree on the fly.
Adaptive Huffman Coding (2)

- Developed based on sibling property
  - A Huffman tree is a binary tree with a weight assigned to every node.
  - Each node in a leaf-end represents a symbol with the weight proportional to its probability.
  - The weight of a node is equal to the sum of its two branches (if existed).
  - Every node can be listed in order or increasing weight from left to right and from leaf to root.

Adaptive Huffman Coding (3)

- Principles:
  - Weight increasing process:
    - Increase the value of weight of a node will cause the tangent in all of this node’s parent nodes.
  - Switching process:
    - After each weight increasing, we need to check the tree satisfies sibling property or not. If against the sibling property, it needs to do the switching process.
    - Switch the node against the property to the most right and up node with the weight less than it.
Adaptive Huffman Coding (4)

- **Initial:** start from a ‘EOF’ with weight 1
- **New node creating example:**
  - create a new node when input a new symbol:
    - assume input a new symbol ‘A’:

```
  EOF
  w=1
  need to create a new node
  EOF
  w=1
  ?
  w=0
  new symbol = ‘A’
  EOF
  w=1
  A
  w=1
```

Please note that, we use the number of occurrence as the weight.

Adaptive Huffman Coding (5)

- **Weight increasing example:**

```
  EOF
  w=1
  B
  w=3
  C
  w=9
  A
  w=3
  EOF
  w=1
  B
  w=3
  received one more ‘B’ changed
  w=5
  C
  w=15
  A
  w=9
```

C.-C. Jay Kuo
Adaptive Huffman Coding (6)

Switching example (I):

- After received two contiguous 'B', switching

Adaptive Huffman Coding (7)

Switching example (II):

- Received one more 'B', switching
Adaptive Huffman Coding (8)

- Applications:
  - lossless image compression
  - text compression
  - audio compression
    - compression ratio can reach up to 1.65:1 for CD-quality audio data (sampled by 44.1kHz for each stereo channel and each sample is represented by 16 bits)
    - can perform better widely from symphonic to folk rock music.

Adaptive Huffman Coding

Encoding Examples

- Encoding
  - Initial adaptive Huffman table as follows

```
Symbol  | Probability |
--------|-------------|
  A     |     1       |
  B     |     1       |
  C     |     1       |
  D     |     1       |
  E     |     1       |
```

```
Initial Huffman Table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
W=1
```

```
W=1
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```

```
W=20
```
Adaptive Huffman Coding

Encoding Examples

- To encode ‘DDDAE’:
  - Consider first ‘D’, encode it as ‘0110’, update the tree.

Switch these two
Adaptive Huffman Coding

Encoding Examples

To encode ‘DDDAE’:

- Consider first ‘D’, encode it as ‘0110’, update the tree.

Switch these two subtrees because 2<3
Adaptive Huffman Coding
Encoding Examples

- To encode ‘DDDAE’:
  - Consider second ‘D’, encode it as ‘010’, then update the tree.

![Diagram of Huffman coding tree]

- To encode ‘DDDAE’:
  - Consider third ‘D’, encode it as ‘011’, then update the tree.

![Diagram of updated Huffman coding tree]
Adaptive Huffman Coding
Encoding Examples

- To encode ‘DDDAE’:
  - Consider third ‘D’, encode it as ‘011’, then update the tree.

```
D 0
W=4
A 1
W=20
```

- To encode ‘DDDAE’:
  - Consider ‘A’, encode it as ‘1’, then update the tree.

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```

```
D 0
W=4
A 1
W=20
```
Adaptive Huffman Coding

Encoding Examples

- To encode ‘DDDAE’:
  - Consider ‘E’, encode it as ‘0101’, then update the tree.

```
  0 1
D 1
  0
D W=4
  0
C W=1
  1
E W=3
B W=3
A W=21
```

Decoding Examples

- Decoding
  - Given symbol set {'A', 'B', 'C', 'D', 'E'}
  - Initial adaptive Huffman table as follows

```
  0 1
R 1
  0
C W=3
  1
D W=1
  0
C W=1
  1
E W=1
```

Adaptive Huffman Coding
Decoding Examples

To decode ‘01001001110100’:
- Go through the table from its root, decode until ‘010’ and get ‘C’. Then update the table.

To decode ‘C01001110100’:
- Go through the table from its root, decode until ‘010’ and get ‘C’. Then update the table.
Adaptive Huffman Coding
Decoding Examples

To decode ‘CC01110100’:
- Go through the table from its root, decode until ‘011’ and get ‘C’. Then update the table.

To decode ‘CCC10100’:
- Go through the table from its root, decode until ‘1’ and get ‘A’. Then update the table.
Adaptive Huffman Coding
Decoding Examples

- To decode ‘CCCA0100’:
  - Go through the table from its root, decode until ‘0100’ and get ‘D’. Then update the table.

![Diagram showing the decoding process]

- Final result:
  - Decoded symbols ‘CCCAD’
  - Huffman table

![Diagram showing the final result]
Arithmetic Coding (1)

- Unlike Huffman code which is a block code (assign variable and integer length code to each block), arithmetic code is a non-block code (also known as tree code).
- Especially useful when dealing with sources with small alphabets, such as binary sources, and alphabets with highly skewed probability.
Arithmetic Coding (2)

- Example: problem of Huffman code:
  \[ \text{Pr}(A)=0.95, \text{Pr}(B)=0.03, \text{Pr}(C)=0.02 \]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
</tbody>
</table>

Entropy=0.335 bits/symbol  
Bavg=1.05 bits/symbol  
Coding Redundancy=0.715 bits/symbol

- Higher-order model still not work:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>A B</td>
<td>111</td>
</tr>
<tr>
<td>A C</td>
<td>100</td>
</tr>
<tr>
<td>B A</td>
<td>1101</td>
</tr>
<tr>
<td>B B</td>
<td>110011</td>
</tr>
<tr>
<td>B C</td>
<td>110001</td>
</tr>
<tr>
<td>C A</td>
<td>101</td>
</tr>
<tr>
<td>C B</td>
<td>110010</td>
</tr>
<tr>
<td>C C</td>
<td>110000</td>
</tr>
</tbody>
</table>

Entropy=0.669 bits/symbol  
Bavg=1.221 bits/symbol  
Coding Redundancy=0.552 bits/symbol

better but still not efficient !!

Always use integer length

Arithmetic Coding (3)

- Example of arithmetic coding (I):
  \[ \text{Pr}(A)=0.2, \text{Pr}(B)=0.4, \text{Pr}(C)=0.2, \text{Pr}(D)=0.1, \text{Pr}(E)=0.1 \]

Example: problem of arithmetic coding (I):

- Example of arithmetic coding (I):
Arithmetic Coding (4)

- Example of arithmetic coding (II):
  - if we want to encode the sequence “BACD”, then any number within the range of [0.2608, 0.2624) can be used to represent this sequence.
  - in general, we choose the central value of the range as our output code, like 0.2616.

Decoding procedure:
- find out the coding interval \([B_L, B_U]\) for the given code \(T\) (in floating) and output the corresponding symbol belong to that interval.

Arithmetic Coding (5)

- change the value of \(T\) by:
  \[ T = \frac{(T - B_L)}{(B_U - B_L)} \]

Decoding example for given code 0.2616:
(1) \(\therefore T = 0.2616 \in [0.2,0.6) \therefore \Rightarrow 'B'\), and \(T = (T - 0.2) / (0.6 - 0.2) = 0.154\)
(2) \(\therefore T = 0.154 \in [0.0,0.2) \therefore \Rightarrow 'A'\), and \(T = (T - 0.0) / (0.2 - 0.0) = 0.77\)
(3) \(\therefore T = 0.77 \in [0.6,0.8) \therefore \Rightarrow 'C'\), and \(T = (T - 0.6) / (0.8 - 0.6) = 0.85\)
Arithmetic Coding (6)

- Adaptive model:
  - forget factor method.
  - higher order model (context model).
- Algorithm for binary arithmetic coder:
  - QM coder by IBM, 1990.
  - Included into CCITT JBIG standard, 1992.

Arithmetic Coding Encoding Examples

- Given symbol set and probability of each symbol, encode ‘DBEFA.’
  
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Assigned range</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (EOF)</td>
<td>0.1</td>
<td>0.0 - 0.1</td>
</tr>
<tr>
<td>B</td>
<td>0.1</td>
<td>0.1 - 0.2</td>
</tr>
<tr>
<td>C</td>
<td>0.1</td>
<td>0.2 - 0.3</td>
</tr>
<tr>
<td>D</td>
<td>0.2</td>
<td>0.3 - 0.5</td>
</tr>
<tr>
<td>E</td>
<td>0.25</td>
<td>0.5 - 0.75</td>
</tr>
<tr>
<td>F</td>
<td>0.25</td>
<td>0.75 - 1.0</td>
</tr>
</tbody>
</table>
Arithmetic Coding

Encoding Examples

- Encode ‘DBEFA’
  - D: \([0.3, 0.5)\)
  - dB: \([0.3 + 0.2 \times 0.1 = 0.32, 0.3 + 0.2 \times 0.2 = 0.34)\)

<table>
<thead>
<tr>
<th></th>
<th>A (EOF)</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>0.0-0.1</td>
<td>0.1-0.2</td>
<td>0.2-0.3</td>
<td>0.3-0.5</td>
<td>0.5-0.75</td>
<td>0.75-1.0</td>
</tr>
</tbody>
</table>

- Encode ‘DBEFA’
  - DBE: \([0.32 + 0.02 \times 0.5 = 0.33, 0.32 + 0.02 \times 0.75 = 0.335)\)
  - DBEF: \([0.33375, 0.335)\)
  - DBEFA: \([0.33375, 0.33875)\)
Arithmetic Coding
Decoding Examples

- Decode ‘0.8302’
- Result: ‘FDBA’

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.0-0.1</th>
<th>0.2</th>
<th>0.3-0.5</th>
<th>0.25</th>
<th>0.75-1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.1</td>
<td>0.0-0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0.1</td>
<td>0.1-0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0.1</td>
<td>0.2-0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0.2</td>
<td>0.3-0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>0.25</td>
<td>0.5-0.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0.25</td>
<td>0.75-1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Contents

- Background
- Huffman coding
- Arithmetic coding
- QM coder
- Lempel Ziv coding
QM Coder (1)

- An adaptive binary arithmetic coder
- An entropy coder standard of JBIG and approved by CCITT in 1992
- A lineal descendant of the Q coder developed by IBM in 1988, but enhanced by the improvements of:
  - interval subdivision
  - probability estimation

QM coder (2)

- Non-binary data need to be modified by binary decision tree

<table>
<thead>
<tr>
<th>DATA</th>
<th>Binary Decision Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1S0</td>
</tr>
<tr>
<td>1</td>
<td>1S10</td>
</tr>
<tr>
<td>2~3</td>
<td>1S110M</td>
</tr>
<tr>
<td>4~7</td>
<td>1S1110MM</td>
</tr>
<tr>
<td>8~15</td>
<td>1S1110MMM</td>
</tr>
</tbody>
</table>
QM Coder (3)

- Basic QM coder algorithm
  - After MPS (Maximum Probability Symbol)
    - $C = C$
    - $A = A(1 - Qe)$
  - After LPS (Least Probability Symbol)
    - $C = C + A(1 - Qe)$
    - $A = A * Qe$

A: Coding interval
C: Coding index, always points to the bottom of A
Qe: Probability of LPS

QM Coder (4)

- Example of basic QM coder algorithm

Input Data Sequence: 0011
Qe=0.25, MPS=0

A0
A1
A2
A3
A4

A2 = A1 x 0.75
A3 = A2 x 0.25
A4 = A3 x 0.25

C4
C3
C2
C1
C0
QM Coder (5)

- Modified QM coder algorithm
  - Potentially unbounded precision for A
    - Left shift 1 bit for A & C whenever A < 0.75 (re-normalization)
    - Generate new Qe after each re-normalization
  - Conditional exchange whenever A < Qe
    - Exchange MPS and LPS
  - Elimination of multiplication
    \[ A \times Q_e = Q_e \text{ if } 15 \geq A \geq 0.75 \]

MPS Re-normalization

Q_e(1) > Q_e(2) > Q_e(3) because of the continuous receiving of MPS
Algorithm of QM Coder

- After MPS
  \[ C = C; \]
  \[ A = A - Q_e; \]
  \[ \text{if (A < 0x8000)} \]
  \[ \{ \]
  \[ \text{if (A < Q_e)} \]
  \[ \{ \]
  \[ C += A; \]
  \[ A = Q_e; \]
  \[ \} \]
  \[ \text{change Q_e state for MPS-renormalization;} \]
  \[ \text{output MSB of } C; \]
  \[ A <<= 1; C <<= 1; \]
  \[ \} \]

- After LPS
  \[ C = C; \]
  \[ A = A - Q_e; \]
  \[ \text{if (A} \geq \text{Q_e)} \]
  \[ \{ \]
  \[ C += A; \]
  \[ A = Q_e; \]
  \[ \} \]
  \[ \text{change Q_e state for LPS-renormalization;} \]
  \[ \text{output MSB of } C; \]
  \[ A <<= 1; C <<= 1; \]

Model of Context of QM Coder

(I) causal context model

(II) parent-children context model in EZW/LZC

K-2 bit plane

K-1 bit plane

K bit plane
Contents

- Background
- Huffman coding
- Arithmetic coding
- QM coder
- Lempel Ziv coding

Lempel Ziv Coding (1)

- Dictionary techniques (I):
  - to keep a list, or dictionary, of the frequently occurring pattern.
  - use a more efficient method to code those pattern in the dictionary.
  - there are two types of dictionary: static and adaptive dictionary. In general, adaptive dictionary performs better than static ones. The well known adaptive cases are: LZ77, LZ78 and LZW.
Lempel Ziv Coding (2)

- Example of dictionary technique:
  - Suppose we have a particular text data consists of 4 characters word (chosen from 32 different characters). For fixed length code case, we need 20 bits ($32^4=2^{20}$) to represent all of the four-character patterns.
  - If we put the 256 most likely four-character in a dictionary, then we only need 8 bits to encode these patterns and use 20 bits to encode those patterns which are not in the dictionary.

Lempel Ziv Coding (3)

- Example of static dictionary:
  assume we have a text data contains ‘a’, ‘b’, ‘c’, ‘d’ and ‘r’ only, then build a static code table as below:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>000</td>
</tr>
<tr>
<td>b</td>
<td>001</td>
</tr>
<tr>
<td>c</td>
<td>010</td>
</tr>
<tr>
<td>d</td>
<td>011</td>
</tr>
<tr>
<td>r</td>
<td>100</td>
</tr>
<tr>
<td>ab</td>
<td>101</td>
</tr>
<tr>
<td>ac</td>
<td>110</td>
</tr>
<tr>
<td>ad</td>
<td>111</td>
</tr>
</tbody>
</table>

coding procedure:
(1) read two characters from the input text data.
(2) if these two characters inside the dictionary, then output the code and goto step (1).
(3) otherwise, output the first character by its code and keep the second character as the first character in next iteration. read only one character from the text and goto step (2).
Most adaptive dictionary based techniques have their roots in two landmark papers by Jacob Ziv and Abraham Lempel in 1977 and 1978. (LZ77/LZ78)

LZW, which developed by Terry Welch “A Technique for High-Performance Data Compression” in IEEE Computer in 1984, make a significant improvement on LZ78.

Algorithm of LZW:
- Initialize the dictionary for each single symbol with an index. Set the buffer to empty.
- A new pattern means the combination of the old pattern in the buffer and the input symbol. if the new pattern is not inside the dictionary, then output the index of the old pattern in the buffer and reset the buffer to the input symbol.
- Otherwise, reset the buffer to the new pattern and no output is necessary.
Lempel Ziv Coding (6)

Example of LZW (I):

(0) assume the text data consists of ‘A’, ‘B’, ‘C’ and ‘D’ only.
initial the dictionary with these four symbols only.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Index</th>
<th>Symbol</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>D</td>
<td>4</td>
</tr>
</tbody>
</table>

we want to encode a text data sequence:
‘ABABBAABBCD’
(1) read ‘A’, already inside our dictionary (index ‘1’), then change the buffer=‘A’

Lempel Ziv Coding (7)

Example of LZW (II):

(2) read ‘B’, but ‘AB’ is not in our dictionary, then output our buffer which is ‘A’ now and add ‘AB’ into our dictionary with index ‘5’. clean the buffer and put ‘B’ into our buffer.
(3) read ‘A’, but ‘BA’ is not in our dictionary, then output the buffer (‘B’) and add ‘BA’ into the dictionary with index ‘6’. reset the buffer and put ‘A’ in it.
(4) read ‘B’ and ‘AB’ is inside the dictionary, no output is necessary but change the value of buffer to ‘AB’.
(5) read ‘B’ and ‘ABB’ is not in the dictionary, then output the buffer which is ‘AB’ (with index ‘5’) and add ‘ABB’ into dictionary with index ‘7’ and buffer change to ‘B’ only.
Lempel Ziv Coding (8)

Example of LZW (III):
(6) read ‘A’ and ‘BA’ is inside the dictionary, no output is necessary but change the value of buffer to ‘BA’.
(7) read ‘A’ but ‘BAA’ is not in our dictionary, then output the buffer (‘BA’) and add ‘BAA’ into the dictionary with index ‘8’, reset the buffer and put ‘A’ in it.
(8) read ‘B’ and ‘AB’ is inside the dictionary, no output is necessary but change the value of buffer to ‘AB’.
(9) read ‘B’ and ‘ABB’ is inside the dictionary, no output is necessary but change the value of buffer to ‘ABB’.
(10) read ‘A’ but ‘ABBA’ is not in our dictionary, then output the buffer (‘ABB’) and add ‘ABAA’ into the dictionary.

Lempel Ziv Coding (9)

Applications:
- “compress” command in UNIX system
- “ARJ”, “PKZIP” and “LHA” in PC system
- telephone network communication standard which used in MODEM, V.42 bis by CCITT
- image compression standard, Graphics Interchange Format (GIF), by Compuserve Information Service
Reference

- Khalid Sayood, “Introduction to Data Compression”, Morgan K Publisher, 1996.