Data Compression

- The Need for Compression
  - One second of uncompressed video (CCIR 601 format)
    - \((720 \times 480 \times Y + 360 \times 480 \times 2 \times U.V.) \times 30\) frames/sec \(\approx 20\) megabytes
  - Two minutes of uncompressed CD-quality music
    - \(44.1\) kHz \(\times 16\) bits/sample \(\times 60 \times 2 \approx 84\) megabits

- Objective
  - To reduce the number of bits required to represent a text file, an image or a video sequence

- Data compression = Modeling + Coding
  - to discover the structures (or redundancy) that exist in the data
    - Modeling
  - to represent the data in a compact form
    - Coding

- Example: variable-length codes
  - The letter "e" appears very often, while "z" is rare in typical English text
    - Assign the shortest code to the letter "e", and longest code to the letter "z"
Digital Communication System

- Encoder (compressor) compress the raw data (original data, or unencoded data) in the input stream and creates an output stream with compressed data (or encoded data)
- Decoder (decompressor) converts in the opposite direction
- Codec = encoder + decoder
- Source coding : coding, data compression
- Channel coding : error control coding

Compression Techniques

- Lossy vs. Lossless Compression
  - Lossless compression
    - no loss of information
    - e.g. text files, medical images, satellite images
  - Lossy compression
    - data cannot be recovered or reconstructed exactly
    - data can be represented with higher compression ratio
    - e.g. speech, images, videos

- Static vs. Adaptive Compression
  - Static method
    - the operations, parameters (or tables) won't be modified in response to the particular original data
  - Semi-adaptive method
    - 1st pass - collect statistics on the data
    - 2nd pass - compress
  - Adaptive method
    - encode based on the statistics of the symbols already encountered
  - Locally adaptive
Compression Techniques

- Symmetrical vs. Asymmetrical Compression
  - Symmetrical method
    - The encoder and decoder use basically the same algorithm but work in “opposite” directions
  - Asymmetrical method
    - The encoder works significantly harder than the decoder
      - The data are compressed once, and will be decompressed and used very often
    - Or the decoder works significantly harder than the encoder
      - The files are updated all the time, and their backup files are not used very often

- Modeling
  - statistical modeling
    - Encode a symbol at a time using the probability of the symbol’s appearance
      - e.g. Morse code, Huffman coding, arithmetic coding
  - Dictionary method
    - Read in input data and look for groups of symbols that appear in a dictionary
      - e.g. Braille code, LZ coding

Measurements of Performance

- Complexity
  - how fast the algorithm performs
- Memory Required
- Amount of Compression
  - Compression ratio
    - The number of bits needed to represent one bit of the original data
      - \( \text{compression ratio} = \frac{\text{size of the compressed data}}{\text{size of the original data}} \)
  - Compression factor
    - \( \text{compression factor} = \frac{\text{size of the original data}}{\text{size of the compressed data}} \)
  - Rate
    - bits per sample (bpp: bits per pixel)
    - bits per second (64Kbps)
- How closely the reconstruction resembles the original (for lossy compression)
  - objective measure
    - SNR (signal-to-noise ratio)
  - subjective measure
    - MOS (Mean Opinion Score)
Intuitive Compression (1)

- Morse Code (1838)
  - Letters are encoded with dots and dashes
    - Shorter sequences are assigned to letters that occur more frequently

Intuitive Compression (2)

- Braille Code (1820s)
  - 3x2 arrays of dots are used to represent text
    - Dots are either raised or flat ⇒ 2^6 = 64 combinations
Example 1 (p.7)

- Source $x_n$
  - 9 11 11 14 15 17 16 17 20 21
  - 5 bits/sample $\times$ 12 $\Rightarrow$ 60 bits

- Model
  - $\hat{x}_n = n + 8$
  - 9 10 11 12 13 14 15 16 17 18 19 20

- Residual (or difference)
  - $e_n = x_n - \hat{x}_n \in \{-1, 0, 1\}$
  - 0 1 0 -1 1 -1 0 1 -1 -1 1 1
  - 2 bits/sample $\times$ 12 $\Rightarrow$ 24 bits
  - 60 - 24 = 36 bits are saved

Example 2 (p.8)

- Source $x_n$
  - 27 28 29 28 26 27 29 28 30 32 34 36 38
  - 6 bits/sample $\times$ 13 $\Rightarrow$ 78 bits

- Model
  - $\hat{x}_n = x_{n-1}$
  - 0 27 28 29 28 26 27 29 28 30 32 34 36

- Residual
  - $e_n = x_n - \hat{x}_n$
  - 27 1 1 -1 -2 1 2 -1 2 2 2 2 2
  - 6 bit for 1st number + 3 bits/sample $\times$ 12 $\Rightarrow$ 42 bits
  - 78 - 42 = 36 bits are saved
Example 3  (p.9)

- **Source**
  - a_barayaran_array_ran_far_faar_faaar_away
  - 8 symbols \{ a, _, b, r, y, n, f, w \}
  - 3 bits/symbol x 41 \rightarrow 123 \text{ bits}

- **Variable-length coding**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency of Occurrence</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>8</td>
<td>000</td>
</tr>
<tr>
<td>_</td>
<td>7</td>
<td>001</td>
</tr>
<tr>
<td>f</td>
<td>3</td>
<td>0100</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
<td>0101</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>0111</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>01100</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>01101</td>
</tr>
</tbody>
</table>

- 106 bits are needed to encode the entire sequence

- 2.56 bits/sample (106 / 41)

---

**Main Problems with VLC**

1. unambiguously decodable
2. minimum average size

---

Mathematical Preliminaries for Lossless Compression

Chapter 2
Information Theory (1)

- Dictionary definitions of information
  - Knowledge derived from study, experience or instruction
  - Knowledge of a specific event or situation; intelligence
  - A collection of facts or data

- Information theory ⇒ to quantify information
  - How much information is included in this piece of data?

- Self-Information
  - event A
  - $P(A)$ is the probability that A will occur
  - self-information of A

\[
i(A) = \log \frac{1}{P(A)} = -\log P(A)
\]

- If the probability of an event is low, the amount of self-information associated with it is low

Information Theory (2)

- Self-Information associated with the occurrence of both event A and B

\[
i(AB) = -\log P(AB)
\]

- for independent events A & B

\[
i(AB) = -\log P(A)P(B) = -\log P(A) - \log P(B) = i(A) + i(B)
\]

- Example
  - A coin toss experiment
    - $P(H) = P(T) = 1/2$
    - If the log base is 2, then
      \[i(H) = i(T) = 1 \text{ bit}\]
    - We need 1 bit to communicate if H happened or not
  - If the coin is not fair
    - $P(H) = 1/8, P(T) = 7/8$
    - $i(H) = 3 \text{ bit}, i(T) = 0.193 \text{ bit}$
    - The occurrence of a HEAD conveys much more information
Information Theory (3)

- **Entropy**
  - a set of independent events \( A_j, j = 1 \sim n \)
  - \( \bigcup A_j = S \)
  - the average self-information (entropy)
    \[
    H(S) = \sum_{i=1}^n P(A_j) \log_2 P(A_j)
    \]
  - \( H(S) \) is largest when all \( n \) probabilities are equal
  - \( H(S) \) is smallest when one \( p_i = 1 \) and all the others =0

- **Redundancy**
  - a set of independent events \( A_j, j = 1 \sim n \)
  - length of the codeword \( l_j, j = 1 \sim n \)
  - redundancy = difference between the entropy and the average length
    \[
    R = \sum_{i=1}^n P(A_j) l_j - H(S)
    \]

Information Theory (4)

- **Example**
  - Source sequence 1 2 1 2 3 3 3 3 1 2 3 3 3 1 2 3 3 3 1 2 3 3 3 1 2 3 3 3 1 2
  - Assume the sequence is independent and identically distributed (iid)
    \[
    P(1) = \frac{5}{20} = 0.25, \quad P(2) = \frac{5}{20} = 0.25, \quad P(3) = \frac{10}{20} = 0.5
    \]
    \[
    H = -(0.25) \log_2 (0.25) - (0.25) \log_2 (0.25) - (0.5) \log_2 (0.5) = 1.5 \text{bits}
    \]
    \[
    \Rightarrow \text{total number of bits} = 1.5 \text{bits/symbol} \times 20 \text{symbol} = 30 \text{bits}
    \]

  - in blocks of two symbols
    \[
    1 2 1 2 3 3 3 3 1 2 3 3 3 3 1 2 3 3 3 1 2 3 3 3 1 2 3 3 3 1 2
    \]
    \[
    P(12) = \frac{5}{10} = 0.5, \quad P(33) = \frac{5}{10} = 0.5
    \]
    \[
    H = -(0.5) \log_2 (0.5) - (0.5) \log_2 (0.5) = 1 \text{bit}
    \]
    \[
    \Rightarrow \text{total number of bits} = 1 \text{bit/block} \times 10 \text{block} = 10 \text{bits}
    \]
Information Theory (5)

- Shannon (1948)

  Given a set of independent events \( A_1, A_2, \ldots, A_n \) with probabilities \( p_j = P(A_j) \)

  The desired properties in the measure of average information \( H \):
  1. \( H \) should be a continuous function of \( p_i \)
     - A small change in \( p_i \) should only cause a small change in \( H \)
  2. If \( p_i = \frac{1}{n} \) for all \( i \), then \( H \) should be a monotonically increasing function of \( n \)
     - If there are more possible outcomes, then more information should be contained in the occurrence of any particular outcome
  3. If the outcome is indicated in multiple stages, the information should be no different than the information associated with the outcome indicated in a single stage

- Shannon showed that only one function can satisfy all these conditions

\[
H = -K \sum_{j=1}^{n} p_j \log_2 p_j
\]

where \( K \) is an arbitrary positive constant

Modeling (p.22)

- Physical Models
  - based on the physics of the data generation process
  - too complicated to understand

- Probability Models
  - For a source \( S \) with alphabet \( A = \{a_1, a_2, \ldots, a_k\} \)
    - the probability model for independent events is given as \( P = \{P(a_1), P(a_2), \ldots, P(a_k)\} \)
    - Only efficient if the assumptions are in accord with reality

- Markov Models
  - \( K \)th-order model
    - the probability of next symbol is dependent on its \( K \) preceding symbols
    \[
P(x_s | x_{s-1}, \ldots, x_{s-K}) = P(x_s | x_{s-1}, \ldots, x_{s-K})
    \]
  - 1st-order model
    \[
P(x_s | x_{s-1}) = P(x_s | x_{s-1}, \ldots, x_{s-K})
    \]
Coding (p.27)

\[ H = - \sum_{j=1}^{i} P(a_j) \log_2 P(a_j) = 1.75 \text{ bits/symbol} \]

<table>
<thead>
<tr>
<th>Letters</th>
<th>P(a)</th>
<th>Code_0</th>
<th>Code_1</th>
<th>Code_2</th>
<th>Code_3</th>
<th>Code_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>0.500</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a_2</td>
<td>0.250</td>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>a_3</td>
<td>0.125</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>011</td>
</tr>
<tr>
<td>a_4</td>
<td>0.125</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td>ave. len.</td>
<td>2.00</td>
<td>1.13</td>
<td>1.25</td>
<td>1.75</td>
<td>1.88</td>
<td></td>
</tr>
</tbody>
</table>

Fixed-length code

ambiguous

Prefix code

not unique decodability

not instantaneous decodability

Coding (Ex. Prefix Codes p.31)

- Prefix Codes
  - No codeword is a prefix to another codeword
  - Uniquely decodable

![Diagram of prefix codes]
The Kraft-McMillan Inequality

Part 1: (McMillan)

Given a uniquely decodable variable-size code \( C \), with \( N \) codes of size \( l_i \), then

\[
K(C) = \sum_{i=1}^{N} 2^{-l_i} \leq 1 \tag{1}
\]

Part 2: (Kraft)

Given a set of \( K \) positive integers \( \{l_1, l_2, \ldots, l_K\} \) that satisfy equation (1), there exists a uniquely decodable variable-size code such that \( l_i \) are the sizes of its individual codes

A code is uniquely decodable \( \iff \) it satisfies relation (1)

Huffman Coding

Chapter 3
Entropy, Coding, Redundancy

- **Entropy H**
  \[ H = - \sum_{j=1}^{k} P(A_j) \log_2 P(A_j) \]
  
  - Entropy is the optimal lower bound on average length of encoded message

- **Redundancy R**
  \[ R = \sum_{j=1}^{k} P(A_j) l_j - H(S) \]
  
  where \( l_j \) is the length of codeword for \( A_j \)

- **Minimum redundancy code**
  - Minimum average code length for a given probability distribution

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Self-Information</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.5</td>
<td>1</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0.25</td>
<td>2</td>
<td>01</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>c</td>
<td>0.125</td>
<td>3</td>
<td>10</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>d</td>
<td>0.125</td>
<td>3</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average length</td>
<td>2</td>
<td>1.75</td>
<td>1.875</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Redundancy</td>
<td>0.25</td>
<td>0</td>
<td>0.125</td>
</tr>
</tbody>
</table>

Shannon-Fano Coding (1)

- **Shannon-Fano coding**
  - Developed by Claude Shannon at Bell Labs and Robert M. Fano at MIT nearly simultaneously

1. Sort the list of symbols in non-increasing probability order
2. Divide the list into two parts with nearly equal total probability
3. Assign 0 to the first half of the list, and 1 to the second half
4. Recursively apply the steps 2 and 3 to each of the two halves, until each list contain only one symbol

The codes are constructed from top to bottom (from the leftmost to rightmost bits)
Shannon-Fano Coding (2)

- Example 1:

<table>
<thead>
<tr>
<th>Letters</th>
<th>$P(a_i)$</th>
<th>$C_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>0.25</td>
<td>00</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.20</td>
<td>01</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.15</td>
<td>100</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.15</td>
<td>101</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.10</td>
<td>110</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.10</td>
<td>1110</td>
</tr>
<tr>
<td>$a_7$</td>
<td>0.05</td>
<td>1111</td>
</tr>
</tbody>
</table>

$$H = - \sum_{j=1}^{7} P(a_j) \log_2 P(a_j)$$
$$= 2.67 \text{ bits/symbol}$$

$$\bar{T} = 0.25 \times 2 + 0.2 \times 2 + 0.15 \times 3 + 0.15 \times 3 + 0.1 \times 4 + 0.05 \times 4$$
$$= 2.7 \text{ bits/symbol}$$

*Example 2:*

<table>
<thead>
<tr>
<th>Letters</th>
<th>$P(a_i)$</th>
<th>$C_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>0.25</td>
<td>00</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.25</td>
<td>01</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.125</td>
<td>100</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.125</td>
<td>101</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.125</td>
<td>110</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.125</td>
<td>111</td>
</tr>
</tbody>
</table>

$$H = 2.5 \text{ bits/symbol}$$

$$\bar{T} = 2.5 \text{ bits/symbol}$$

The best results are produced when $P(a_i)$ are negative power of 2

Huffman Coding (1)

- Optimum prefix codes
  - Symbols that have a higher probability of occurrence will have shorter codewords than symbols that occur less frequently
  - The two symbols that occur least frequently will have the same length
    - Their codewords differ only in the last bit

- Huffman coding
  - Construct the binary tree
    - Sort the list of symbols in non-increasing probability order
    - Start at the leaf nodes, which represents the source symbols
    - Create a new node for the two symbols with smallest probabilities
    - This node is assigned the probability equal to the sum of the probabilities of the two child nodes
    - Add this node to the list of symbols, and remove the two child nodes from the list
    - Repeat the previous steps until only one single node is left. This node is designated the root of the tree.
  - Obtain the code for each symbol
    - Traverse the tree from the root to each leaf node and assign a code 0 to the left branch and code 1 to a right branch

Developed by David Huffman (1951) as part of a class assignment
Huffman Coding (2)  

- Example:

<table>
<thead>
<tr>
<th>Letters</th>
<th>$P(a_i)$</th>
<th>Code 1</th>
<th>Code 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>0.4</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.2</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.2</td>
<td>000</td>
<td>11</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.1</td>
<td>0010</td>
<td>010</td>
</tr>
<tr>
<td>$a_c$</td>
<td>0.1</td>
<td>0011</td>
<td>011</td>
</tr>
</tbody>
</table>

  In descending order

  The codes are constructed from the bottom up (from the rightmost to leftmost bits)

  Minimum variance Huffman code

  $H = 2.122$ bits/symbol

  $I_4 = I_5 = 2.2$ bits/symbol

  $\sigma_i^2 = 0.4(1-2.2)^2 + 0.2(2-2.2)^2 + 0.2(3-2.2)^2 + 0.1(4-2.2)^2$

  $= 0.16$

  Always put the combined letter as high in the list as possible

Huffman Coding (3)  

- Code 1

- Code 2 (minimum variance Huffman code)

  $\Phi$ average code length = sum of the internal nodes

  $0.2 = 0.1 + 0.1$

  $0.4 = 0.2 + 0.2 = 0.2 + 0.1 + 0.1$

  $0.6 = 0.4 + 0.2 = 0.2 + 0.1 + 0.1 + 0.2$

  $1 = 0.6 + 0.4 = 0.2 + 0.1 + 0.1 + 0.2 + 0.4$

  $2.2 = 0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = 7$
Length of Huffman Codes

For a source $S$ with alphabet $A = \{a_1, a_2, \ldots, a_n\}$ and probability model $\{P(a_1), P(a_2), \ldots, P(a_n)\}$

\[ \Rightarrow \text{The average code length } \bar{T} \text{ is bounded by } \]

\[ H(S) \leq \bar{T} < H(S) + 1 \]

$H(S)$ : The entropy for a source $S$

Extended Huffman Codes

- Extended Huffman Codes
  - generate one codeword for every $n$ symbols
  - the source $S^{(n)}$ with alphabet $A^{(n)} = \{a_1a_1 \ldots a_1a_1, a_2 \ldots a_n, a_{n+1} \ldots a_{2n}, \ldots, a_{m-1}a_m \ldots a_m \}$

- Let $\bar{T^{(n)}}$ denote the average code length for the source $S^{(n)}$

\[ H(S^{(n)}) \leq \bar{T^{(n)}} < H(S^{(n)}) + 1 \]

\[ \therefore \frac{1}{n} \bar{T^{(n)}} \quad \& \quad H(S^{(n)}) = -\sum_{i_1} P(a_{i_1}) \ldots \sum_{i_{\frac{m}{n}}} P(a_{i_{\frac{m}{n}}}) \log_2 P(a_{i_1}) \ldots P(a_{i_{\frac{m}{n}}}) \]
\[ = \ldots \]
\[ = -\sum_{i} P(a_{i_1}) \log_2 P(a_{i_1}) \ldots - \sum_{i_{\frac{m}{n}}} P(a_{i_{\frac{m}{n}}}) \log_2 P(a_{i_{\frac{m}{n}}}) \]

\[ = H(S) \leq \bar{T} < H(S) + \frac{1}{n} \]
Adaptive Huffman Coding (1)

- To develop the Huffman code based on the statistics of the symbols already encountered
  - Faller (1973), Gallagher (1978), Knuth (1985) \(\rightarrow\) FGK algorithm
  - Vitter (1987) \(\rightarrow\) V algorithm

Adaptive Huffman Coding (2)

- Sibling property
  - A binary tree with \(n\) leaves of nonnegative weight is a Huffman tree if and only if
  1. the weight of each internal node equals the sum of the weights of its children
  2. the nodes can be numbered in non-decreasing order by weight,
     
     weight: \(x_1 \leq x_2 \leq \ldots \leq x_{2^n-1}\)
     
     node number: \(y_1, y_2, \ldots, y_{2^n-1}\)

     so that nodes \(y_{2^i-1}\) and \(y_{2^j}\) are siblings (for \(1 \leq j \leq n-1\)) and their common parent node is higher in the numbering

- Updating the tree
  - both encoder and decoder start with the same tree
  - process next symbol
  - both encoder and decoder modifying the tree using sibling property
Adaptive Huffman Coding (3)

- NYT (not yet transmitted) code
  - weight = 0
  - each uncompressed symbol is preceded by a variable-size NYT code
  - the smallest node number is assigned to NYT

- Uncompressed code
  - for a source alphabet of size \( m \)

\[
m = 2^r + r, \quad 0 \leq r < 2^e
\]

\[
a_k = \begin{cases} 
  k - 1 \text{ in } (e+1) \text{ bit} & 1 \leq k \leq 2^r \\
  k - r - 1 \text{ in } e \text{ bit} & 2^r < k \leq m 
\end{cases}
\]

Example:

\[
m = 26 = 2^4 + 10 \\
\begin{array}{|c|c|c|}
\hline
k & \text{symbols} & \text{code} \\
\hline
1 & a & 00000 \\
2 & b & 00001 \\
3 & c & 00010 \\
4 & d & 00011 \\
5 & : & : \\
10 & : & : \\
11 & : & : \\
19 & s & 10010 \\
20 & t & 10011 \\
21 & u & 10100 \\
22 & v & 10101 \\
23 & : & : \\
24 & : & : \\
25 & : & : \\
26 & z & 11111 \\
\hline
\end{array}
\]

Adaptive Huffman Coding (4)

- Example:
  - source alphabet of size \( m = 26 \) \{a, b, ... z\}
  - input: a a r d v ...

- output: 00000 a NYT r

\[
\begin{array}{ccc}
\text{NYT} & 0 & \text{NYT} \\
0 & 51 & 0 & 49 \\
\text{(a)} & 1 & 50 & 2 & 50 \\
\text{(aa)} & 2 & 49 & 1 & 50 \\
\text{(aaa)} & 3 & 47 & 1 & 48 \\
\text{(aaa)} & 4 & 46 & 1 & 47 \\
\end{array}
\]

17 / 32
Adaptive Huffman Coding (5)

- Example:
  - source alphabet of size $m=26\{a,b,...,z\}$
  - input: a a r d v ...

```
a  a  NYT r  NYT d  NYT v
- output: 00000 1 0 10001 00 00011 000 1011
```

Adaptive Huffman Coding (6)

- Example:
  - source alphabet of size $m=26\{a,b,...,z\}$
  - input: a a r d v ...

```
a  a  r  d  v
- output: 00000 1 0 10001 00 00011 000 1011
```
Tunstall Codes

- Tunstall codes
  - Variable-length symbols $\rightarrow$ Fixed-length codes

- Example: 3-bit code
  - Source alphabet \{a, b, c\}

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.6</td>
</tr>
<tr>
<td>b</td>
<td>0.3</td>
</tr>
<tr>
<td>c</td>
<td>0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0.3</td>
</tr>
<tr>
<td>c</td>
<td>0.1</td>
</tr>
<tr>
<td>ab</td>
<td>0.18</td>
</tr>
<tr>
<td>ac</td>
<td>0.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Probability</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaaa</td>
<td>0.216</td>
<td>100</td>
</tr>
<tr>
<td>aab</td>
<td>0.108</td>
<td>101</td>
</tr>
<tr>
<td>aac</td>
<td>0.36</td>
<td>110</td>
</tr>
<tr>
<td>ab</td>
<td>0.18</td>
<td>010</td>
</tr>
<tr>
<td>sc</td>
<td>0.06</td>
<td>011</td>
</tr>
<tr>
<td>ab</td>
<td>0.18</td>
<td>010</td>
</tr>
<tr>
<td>ac</td>
<td>0.06</td>
<td>011</td>
</tr>
<tr>
<td>aaaa</td>
<td>0.216</td>
<td>100</td>
</tr>
<tr>
<td>aab</td>
<td>0.108</td>
<td>101</td>
</tr>
<tr>
<td>aac</td>
<td>0.36</td>
<td>110</td>
</tr>
<tr>
<td>a</td>
<td>0.6</td>
<td>00</td>
</tr>
<tr>
<td>aab</td>
<td>0.108</td>
<td>101</td>
</tr>
<tr>
<td>aac</td>
<td>0.36</td>
<td>110</td>
</tr>
</tbody>
</table>

Arithmetic Coding

Chapter 4
Example 1

For a source alphabet $S$ with highly skewed probabilities

1. Assign one codeword for each symbol

<table>
<thead>
<tr>
<th>Letters</th>
<th>$P(a_i)$</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>0.99</td>
<td>0</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.01</td>
<td>1</td>
</tr>
</tbody>
</table>

$H = 0.08$ bits/symbol
$T = 1$ bits/symbol

2. Assign one codeword for every two symbols

<table>
<thead>
<tr>
<th>Letters</th>
<th>$P(a_i)$</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1a_1$</td>
<td>0.9801</td>
<td>0</td>
</tr>
<tr>
<td>$a_1a_2$</td>
<td>0.0099</td>
<td>11</td>
</tr>
<tr>
<td>$a_1a_1$</td>
<td>0.0099</td>
<td>100</td>
</tr>
<tr>
<td>$a_1a_3$</td>
<td>0.0009</td>
<td>101.00</td>
</tr>
</tbody>
</table>

$T^{(2)} = 1.0299$ bits/block
$T = \frac{T^{(2)}}{2} = 0.51495$ bits/symbol

Example 2

For a source alphabet $S$ with highly skewed probabilities

1. Assign one codeword for every three symbols

<table>
<thead>
<tr>
<th>Letters</th>
<th>$P(a_i)$</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1a_1a_1$</td>
<td>0.970299</td>
<td>0</td>
</tr>
<tr>
<td>$a_1a_1a_2$</td>
<td>0.009801</td>
<td>100</td>
</tr>
<tr>
<td>$a_1a_1a_3$</td>
<td>0.009801</td>
<td>101</td>
</tr>
<tr>
<td>$a_1a_2a_2$</td>
<td>0.000999</td>
<td>11100</td>
</tr>
<tr>
<td>$a_1a_3a_1$</td>
<td>0.009801</td>
<td>110</td>
</tr>
<tr>
<td>$a_1a_3a_2$</td>
<td>0.000999</td>
<td>11101</td>
</tr>
<tr>
<td>$a_2a_1a_1$</td>
<td>0.000999</td>
<td>11110</td>
</tr>
<tr>
<td>$a_2a_1a_2$</td>
<td>0.000999</td>
<td>11111</td>
</tr>
</tbody>
</table>

$H = 0.08$ bits/symbol
$T^{(3)} = 1.05998$ bits/block
$T = \frac{T^{(3)}}{3} = 0.3533$ bits/symbol

To "block" several symbols together at a time
per-symbol inefficiency is now spread over the whole block
the size of the codebook increases exponentially
e.g. $A = \{a_1, a_2, \ldots, a_n\}$
to generate one codeword for every $n$ symbols
$\Rightarrow m^n$ codewords are needed.
Arithmetic Coding (1)

- Arithmetic code
  - assign codewords to individual symbols
  - assign one (normally long) code to the entire input stream
  - A source message is represented by an interval of real numbers in [0,1)

- Example:

<table>
<thead>
<tr>
<th>Letters</th>
<th>Probability $P(a_i)$</th>
<th>cdf $F_i(i)$</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>0.8</td>
<td>0.8</td>
<td>[0,0.8)</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.02</td>
<td>0.82</td>
<td>[0.8,0.82]</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.18</td>
<td>1</td>
<td>[0.82,1)</td>
</tr>
</tbody>
</table>

Cumulative distribution function (cdf)

$$F_i(i) = \sum_{k=1}^{i} P(X = k) \quad \text{or} \quad \sum_{k=1}^{i} P(a_k)$$

Arithmetic Coding (2)

- Input sequence: $X = (x_1, x_2, ..., x_n) = (a_1, a_2, a_3, a_4)$

![Diagram of arithmetic coding process]
**Arithmetic Coding (3)**

Code = 0.772352

\[
\begin{align*}
X_1 &= a_1 \\
G_1 &= 0.656 \\
U_1 &= 0.8
\end{align*}
\]

Decoding steps:
1. **Initialization**
   - \( X_0 = 0 \)
   - \( G_0 = 1 \)

2. For each \( k \),
   - \[ code^\prime = \frac{code - T_k}{U_k - L_k} = \frac{0.772352 - 0.656}{0.8 - 0.656} = 0.808 \]
   - \( x_k = 0.82 \)
   - \[ G_k = 0.656 \\
   U_k = 0.8 \]

3. Find \( x_k \) s.t.
   - \( F_k(x_k - 1) \leq code^\prime < F_k(x_k) \)

4. Update \( I_k \) & \( U_k \)

5. Continue until the entire sequence has been decoded

---

**Arithmetic Coding (4)**

- How to know when the entire sequence has been decoded?
  - The length of the sequence is known in advance
    - the encoder can start by writing the unencoded size on the output stream
    - How many bits should be preserved
  - An additional symbol EOF is added
    - with a small probability

- A high-probability symbol narrows the interval less than a low-probability does
  - high-probability contribute fewer bits to the coded sequence

<table>
<thead>
<tr>
<th>Letters</th>
<th>Probability ( P(a_i) )</th>
<th>CDF ( F_k(i) )</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>0.8</td>
<td>0.8</td>
<td>([0, 0.8))</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>0.02</td>
<td>0.82</td>
<td>([0.8, 0.82))</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>0.18</td>
<td>1</td>
<td>([0.82, 1))</td>
</tr>
</tbody>
</table>
Adaptive Arithmetic Coding

- Example:
  - for a source alphabet \{a,b,c\}
  - input sequence = b c c b

  ![Diagram](Image)

Implementation

- Rescaling the interval
  - the precision required to represent the interval grows with the length of the sequence
  - once the codec knows which portion contains the code
    - ignore the portion which not containing the code
    - concentrate on the portion contains the code
  - map the interval containing the code to the full [0,1) interval

- Incremental encoding and decoding
  - to generate portions of the code as the sequence is being observed, rather than wait until the entire sequence has been observed before transmitting the first bit

- Integer arithmetic
  - to prevent loss of precision due to roundoff error
  - to speed up arithmetic coding
Rescaling & Incremental Coding

- Incremental encoding
  - send the MSB when $r_i$ and $u_i$ have a common prefix
- Rescaling the interval
  - map the half interval containing the code to the $[0, 1)$

If range $= [0, 0.5) = [0.0xx..., 0.0xx...]_2$
send the binary code = 0
left shift

$E_1: [0,0.5) \rightarrow [0,1)$
$E_1(x) = 2x$

If range $= [0.5, 1) = [0.1xx..., 0.1xx...]_2$
send the binary code = 1
left shift

$E_2: [0.5,1) \rightarrow [0,1)$
$E_2(x) = 2(x-0.5)$

Incremental Encoding

- Example: input sequence $= (a_1, a_2, a_3)$

<table>
<thead>
<tr>
<th>Letters</th>
<th>$P(a_i)$</th>
<th>$F(x)$</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>0.8</td>
<td>0.8</td>
<td>[0,0.8]</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.02</td>
<td>0.82</td>
<td>[0.8,0.82]</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.18</td>
<td>1</td>
<td>[0.82,1]</td>
</tr>
</tbody>
</table>

Let $r_i$ be a real number in the range $[0,1)$ and $u_i$ be the upper bound of the range.

Output code: 11000110...0

($0.1100011_2 = 0.7734375$)
Incremental Decoding

- Example: input bit stream $1100110...0$
  - to decode the first symbol unambiguously

  # bits needed for the code value $p = \lceil -\log_{2}(\text{smallest interval}) \rceil = \lceil -\log_{2}(0.02) \rceil = \lceil 7.64 \rceil = 6$

Incremental Coding (1)

- If $\text{range} \in [0.xxx..., 0.xxx...]_2$
  
  $E_1: [0, 0.5) \rightarrow [0, 1)$
  
  $E_1(x) = 2x$

- If $\text{range} \in [0.xxx..., 1.xxx...]_2$
  
  $E_2: [0.5, 1) \rightarrow [0, 1)$
  
  $E_2(x) = 2(x - 0.5)$

- If $\text{range} \in [0.xxx..., 1.xxx...]_2$

- $E_3: [0.25, 0.75) \rightarrow [0.1)$
  
  $E_3(x) = 2(x - 0.25)$
Incremental Coding (2)

- $E_3 E_2$
  - send code 1 0

- $E_3 E_3 \ldots E_3 E_2$
  - send code $1 \ 0 \ 0 \ldots 0$

Example:

$$\left[ \frac{1}{2} \ 2 \frac{3}{4} \right]$$

code = .1000

Incremental Coding (3)

- $E_3 E_1$
  - send code 0 1

- $E_3 E_3 \ldots E_3 E_1$
  - send code $0 \ 1 \ 1 \ldots 1$

Example:

$$\left[ \frac{1}{4} \frac{3}{2} \right]$$

code = .0111
Incremental Coding (4)

- Example: input sequence = \((a_1, a_2, a_3, a_4)\)

\[
\begin{align*}
\hat{p}^{(0)} &= 0 \\
\hat{u}^{(0)} &= 1 \\
\hat{p}^{(1)} &= 0.656 \\
\hat{u}^{(1)} &= 0.8 \\
\hat{p}^{(2)} &= 0.5848 \\
\hat{u}^{(2)} &= 0.59632 \\
\end{align*}
\]

send code = 10

(next counter = 0)

\[
\begin{align*}
\hat{p}^{(3)} &= 2 \times (0.656 - 0.5) - 0.312 \\
\hat{u}^{(3)} &= 2 \times (0.8 - 0.5) - 0.6 \\
\end{align*}
\]

(counter = 1)

\[
\begin{align*}
\hat{p}^{(3)} &= 0.3392 \\
\hat{u}^{(3)} &= 0.38528 \\
\end{align*}
\]

send code = 1

(use value = 0.5)

send code = 10

Output code 1100110...0

---

Incremental Decoding

Example: input bit stream 11000110...0

# bits needed for the code value = \([-\log_{10}(\text{smallest interval})]\) = \([-\log_{10}(0.02)]\) = 5.64 = 6

\[
\begin{align*}
\hat{p}^{(0)} &= 0 \\
\hat{u}^{(0)} &= 1 \\
\hat{p}^{(1)} &= 0.656 \\
\hat{u}^{(1)} &= 0.8 \\
\hat{p}^{(2)} &= 0.5848 \\
\hat{u}^{(2)} &= 0.59632 \\
\hat{p}^{(3)} &= 0.3392 \\
\hat{u}^{(3)} &= 0.38528 \\
\hat{p}^{(4)} &= 0.3568 \\
\hat{u}^{(4)} &= 0.54112 \\
\end{align*}
\]

(counter = 1)

Decode \(a_1, a_2, a_3, a_4\)
Integer Arithmetic (1)

- Map \([0,1)\) to the range of \(2^m\) binary word

\[
0 \rightarrow \underbrace{00\ldots0}_m
\]

- Cumulative probability

\[
F_k(x) = \frac{\text{Cum\_Count}(k)}{\text{Total\_Count}}
\]

\[
\text{Cum\_Count}(k) = \sum_{i=1}^{k} n_i
\]

- Interval updating

\[
\begin{align*}
\left[ f^{(n)} = f^{(n-1)} + \left[ n^{(n-1)} - f^{(n-1)} + 1 \right] F_k(x_n - 1) \right] \\
\left[ u^{(n)} = f^{(n)} + \left[ n^{(n-1)} - f^{(n-1)} + 1 \right] F_k(x_n) \right] - 1
\end{align*}
\]


Integer Arithmetic (2)

- Example (m=8): input sequence = \((a_1, a_2, a_3)\)

\[
\begin{align*}
(1) &= 0 \rightarrow (00000000)_2 \\
(2) &= 255 \rightarrow (11111111)_2 \\
(3) &= 167 \rightarrow (101001111)_2 \\
(4) &= 203 \rightarrow (110010111)_2
\end{align*}
\]

\[
\text{send code} = 1
\]

\[
\begin{align*}
(5) &= 0 \rightarrow (00000000)_2 \\
(6) &= 203 \rightarrow (110010111)_2
\end{align*}
\]

\[
\text{ed code} = 0
\]

\[
\begin{align*}
(7) &= 146 \rightarrow (1010010010)_2 \\
(8) &= 148 \rightarrow (1010010100)_2
\end{align*}
\]

\[
\text{send code} = 0
\]

\[
\begin{align*}
(9) &= 0 \rightarrow (00000000)_2 \\
(10) &= 191 \rightarrow (1011110011)_2
\end{align*}
\]

\[
\text{ed code} = 0
\]

\[
\begin{align*}
(11) &= 0 \rightarrow (00000000)_2 \\
(12) &= 152 \rightarrow (1001100000)_2
\end{align*}
\]

\[
\text{use value 0}
\]

\[
\text{send code} = 0
\]

Output code

\[
110001001000000000
\]
Dictionary Techniques

Chapter 5

- Split the input into two classes
  - Frequently occurring patterns
    - Encoded with a reference to a dictionary
  - Infrequently occurring patterns
    - Encoded using less efficient method

- Dictionary methods
  - Build a dictionary of frequently occurring patterns
  - Encode input patterns as an index in the dictionary
  - The size of the dictionary must be much smaller than the number of all possible patterns
  - Useful with sources that generate a relatively small number of patterns quite frequently, such as text sources and computer commands
Static Dictionary

- Example
  - For a five-letter alphabet \( A = \{a, b, c, d, r\} \)
  - To encode the sequence: abracadabra

\[
\begin{array}{ccc}
ab & r & ac & ad & ab & r & a \\
101 & 100 & 110 & 111 & 101 & 100 & 000 \\
\end{array}
\]

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

Adaptive Dictionary

- Adaptive dictionary
  - Adapt to the characteristics of the source
  - start with an empty dictionary
  - add entries as they are found in the input stream

- LZ77

- LZSS

- LZ78

- LZW
LZ77, LZ78

- LZ77
  - The dictionary is a portion of the previously encoded sequence

- LZ78
  - Keep an explicit dictionary of previously occurring patterns
    - `encode(Index, New_symbol)`

Example

<table>
<thead>
<tr>
<th>Encoder Output</th>
<th>Index</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 'w')</td>
<td>1</td>
<td>w</td>
</tr>
<tr>
<td>(0, 'a')</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>(0, 'b')</td>
<td>3</td>
<td>b</td>
</tr>
<tr>
<td>(3, 'a')</td>
<td>4</td>
<td>ba</td>
</tr>
<tr>
<td>(0, 'c')</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>(1, 'a')</td>
<td>6</td>
<td>wa</td>
</tr>
<tr>
<td>(3, 'b')</td>
<td>7</td>
<td>bo</td>
</tr>
<tr>
<td>(2, 'c')</td>
<td>8</td>
<td>ac</td>
</tr>
<tr>
<td>(6, 'b')</td>
<td>9</td>
<td>wab</td>
</tr>
</tbody>
</table>

Input sequence:

```
waiba waiba waiba waiba waiba waiba
```
Prediction Coding

- Example: transform the input sequence into another sequence that has a more skewed set of probabilities
  - Source $x_n$
    - 1 2 5 7 2 -2 0 -5 -3 -1 1 -2 -7 -4 -2 1 3 4
    - $H = 3.53$ bits
  - Model
    - $\hat{x}_n = x_{n-4} + 2$
    - Predicted value: 0 3 4 7 9 4 0 2 -3 -1 1 3 0 -5 -2 0 3 5
  - Residual (or difference)
    - $e_n = x_n - \hat{x}_n$
    - Residual sequence: 1 -1 1 0 -7 -6 0 -7 0 0 0 -5 -7 1 0 1 0 -1
    - $H = 2.26$ bits

Lossless Compression

- Dictionary based method
  - Break the data into frequently occurring patterns that are saved in a dictionary
  - Encode the input patterns as an index in the dictionary
- Statistical method
  - Modeling
    - Static or adaptive
    - Assign probabilities to the input symbols
  - Coding
    - Code the symbols based on those probabilities
- Most models are based on one of the two approaches
  - Frequency
    - The model assigns probabilities to the input symbol based on their frequencies of occurrence
  - Context
    - The model considers the context of a symbol when assigning it a probability