CSE 521
Algorithms
Spring 2003

Entropy
Arithmetic Coding

Why Compress

- **Conserve storage space**
- **Reduce time for transmission**
  - Faster to encode, send, then decode than to send the original
- **Progressive transmission**
  - Some compression techniques allow us to send the most important bits first so we can get a low-resolution version of some data before getting the high-fidelity version
- **Reduce computation**
  - Use less data to achieve an approximate answer

Braille

- System to read text by feeling raised dots on paper (or on electronic displays). Invented in 1820s by Louis Braille, a French blind man.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>th</td>
<td>ch</td>
<td>gh</td>
<td></td>
</tr>
</tbody>
</table>

Braille Example

Clear text:
Call me Ishmael. Some years ago -- never mind how long precisely -- having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. (238 characters)

Grade 2 Braille in ASCII.
call me j%maeL4 `'s ye$>g$ ago -- n"e m9d h l k
precisely -- h轩辕 | l L no m" oy 9 my purse! l & no?+
picu/$g $ 9)er'e me on 1%ore1 \J j $g$ $ j wd sail
ab a L l & see l watly "p (11_wd (203 characters)

Compression ratio = 238/203 = 1.17

Lossless Compression

- Data is not lost - the original is really needed.
  - text compression
  - compression of computer binaries to fit on a floppy
- Compression ratio typically no better than 4:1 for lossless compression on many kinds of files.

- **Statistical Techniques**
  - Huffman coding
  - Arithmetic coding
  - Golomb coding
- **Dictionary techniques**
  - LZW, LZ77
  - Sequitur
  - Burrows-Wheeler Method
- **Standards** - Morse code, Braille, Unix compress, gzip, zip, bzip, GIF, JPEG, Lossless JPEG
Why is Data Compression Possible

- Most data from nature has redundancy
  - There is more data than the actual information contained in the data.
  - Squeezing out the excess data amounts to compression.
  - However, unsqueezing out is necessary to be able to figure out what the data means.
- **Information theory** is needed to understand the limits of compression and give clues on how to compress well.

Information Theory

- Developed by Shannon in the 1940’s and 50’s
- Attempts to explain the limits of communication using probability theory.
- Example: Suppose English text is being sent
  - Suppose a “t” is received. Given English, the next symbol being a “z” has very low probability, the next symbol being a “h” has much higher probability. Receiving a “z” has much more information in it than receiving a “h”. We already knew it was more likely we would receive an “h”.

First-order Information

- Suppose we are given symbols \( \{a_1, a_2, \ldots, a_m\} \).
- \( P(a_i) \) = probability of symbol \( a_i \) occurring in the absence of any other information.
  - \( P(a_1) + P(a_2) + \ldots + P(a_m) = 1 \)
- \( \inf(a_i) = -\log_2 P(a_i) \) bits is the information of \( a_i \) in bits.

Example

- \( \{a, b, c\} \) with \( P(a) = 1/8 \), \( P(b) = 1/4 \), \( P(c) = 5/8 \)
  - \( \inf(a) = -\log_2(1/8) = 3 \)
  - \( \inf(b) = -\log_2(1/4) = 2 \)
  - \( \inf(c) = -\log_2(5/8) = .678 \)
  - Receiving an “a” has more information than receiving a “b” or “c”.

First Order Entropy

- The first order entropy is defined for a probability distribution over symbols \( \{a_1, a_2, \ldots, a_m\} \).
  - \( H = -\sum_{i=1}^{m} P(a_i) \log_2(P(a_i)) \)
- \( H \) is the average number of bits required to code up a symbol, given all we know is the probability distribution of the symbols.
- \( H \) is the Shannon lower bound on the average number of bits to code a symbol in this “source model”.
- Stronger models of entropy include context. We’ll talk about this later.

Entropy Examples

- \( \{a, b, c\} \) with a 1/8, b 1/4, c 5/8.
  - \( H = 1/8 *3 + 1/4 * 2 + 5/8 * .678 = 1.3 \) bits/symbol
- \( \{a, b, c\} \) with a 1/3, b 1/3, c 1/3. (worst case)
  - \( H = 3 * (1/3) * \log_2(1/3) = 1.6 \) bits/symbol
- \( \{a, b, c\} \) with a 1, b 0, c 0 (best case)
  - \( H = -1 * \log_2(1) = 0 \)
  - Note that the standard coding of 3 symbols takes 2 bits.
**Entropy Curve**

- Suppose we have two symbols with probabilities $x$ and $1-x$, respectively.

![Entropy Curve Graph]

- Maximum entropy at $0.5$.

\[-x \log x + (1-x) \log (1-x)\]

**Reals in Binary**

- Any real number $x$ in the interval $[0,1)$ can be represented in binary as $b_1b_2\ldots$, where $b_i$ is a bit.

![Binary Representation]

**First Conversion**

- $L := 0$; $R := 1$; $i := 1$
- while $x > L$ *
  - if $x < (L+R)/2$ then $b_i := 0$ ; $R := (L+R)/2$;
  - if $x > (L+R)/2$ then $b_i := 1$ ; $L := (L+R)/2$;
- $i := i + 1$
- end { while }
- $b_j := 0$ for all $j \geq i$

* Invariant: $x$ is always in the interval $[L,R)$

**Conversion using Scaling**

- Always scale the interval to unit size, but $x$ must be changed as part of the scaling.

**Binary Conversion with Scaling**

- $y := x$; $i := 0$
- while $y > 0$ *
  - $i := i + 1$
  - if $y < 1/2$ then $b_i := 0$; $y := 2y$;
  - if $y \geq 1/2$ then $b_i := 1$; $y := 2y - 1$;
- end { while }
- $b_j := 0$ for all $j \geq i + 1$

* Invariant: $x = .b_1b_2\ldots b_i + y/2^i$
Arithmetic Coding

Basic idea in arithmetic coding:
- represent each string \( x \) of length \( n \) by a unique interval \([L, R)\) in \([0,1)\).
- The width \( r = R - L \) of the interval \([L, R)\) represents the probability of \( x \) occurring.
- The interval \([L, R)\) can itself be represented by any number, called a tag, within the half open interval.
- The \( k \) significant bits of the tag \( t_1, t_2, ..., t_k \) is the code of \( x \). That is, \( t_1, t_2, ..., t_k, 000... \) is in the interval \([L, R)\).

Some Tags are Better than Others

- For fixed length strings provides a good prefix code.
- Choose \( k \) to be as small as possible so that \( L < t_1, t_2, t_3, ..., 000... < R \).

Code Generation from Tag

- If binary tag is \( t_1, t_2, t_3, ..., = (L+R)/2 \) in \([L, R)\) then we want to choose \( k \) to form the code \( t_1, t_2, ..., t_k \).
- Short code:
  - \( L \leq t_1, t_2, ..., t_k, 000... < R \).
- Guaranteed code:
  - \( L \leq t_1, t_2, ..., b_1, b_2, b_3, ..., < R \) for any bits \( b_1, b_2, b_3, ... \).
  - \( \log \left( 1/(R - L) \right) \) + 1
  - For example: \( 00000000, ..., 000001010... \), tag = \( .000001001... \)
  - Short code: 0
  - Guaranteed code: 000001

Example of Arithmetic Coding (1)

- 1. tag must be in the half open interval.
- 2. tag can be chosen to be \((L+R)/2\).
- 3. code is the significant bits of the tag.

Example of Codes

- \( P(a) = 1/3, P(b) = 2/3 \).
- \( \text{tag} = (L+R)/2 \)
- \( \text{code} = 0111 \)
- Alternative tag = 14/37 = .100001001...
- \( \text{code} = 1 \)

Guaranteed Code Example

- \( P(a) = 1/3, P(b) = 2/3 \).
- \( \text{tag} = (L+R)/2 \)
- \( \text{short code} \)
- \( \text{Prefix code} \)
### Arithmetic Coding Algorithm

- \( P(a_1), P(a_2), \ldots, P(a_m) \)
- \( C(a) = P(a_1) + P(a_2) + \ldots + P(a_i) \)
- Encode \( x_1x_2\ldots x_n \)

1. Initialize \( L := 0 \) and \( R := 1 \);
2. For \( i = 1 \) to \( n \) do:
   1. \( W := R - L \);
   2. \( L := L + W \cdot C(x_i) \);
   3. \( R := L + W \cdot P(x_i) \);
   4. \( t := (L + R)/2 \);
   5. Choose code for the tag

### Arithmetic Coding Example

- \( P(a) = 1/4, P(b) = 1/2, P(c) = 1/4 \)
- \( C(a) = 0, C(b) = 1/4, C(c) = 3/4 \)
- abc

<table>
<thead>
<tr>
<th>Symbol</th>
<th>( W )</th>
<th>( L )</th>
<th>( R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1/2</td>
<td>0</td>
<td>1/2</td>
</tr>
<tr>
<td>b</td>
<td>1/2</td>
<td>1/4</td>
<td>3/4</td>
</tr>
<tr>
<td>c</td>
<td>1/2</td>
<td>3/8</td>
<td>7/8</td>
</tr>
</tbody>
</table>

- \( a \)
  - \( W := R - L \);
  - \( L := L + W \cdot C(a) \);
  - \( R := L + W \cdot P(a) \);

- \( \text{choose code for the tag} \)

---

### Decoding (1)

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...

```
.0001000...  output a
.a

CSE 521 - Arithmetic Coding - Spring 2003 27
```

### Decoding (2)

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...

```
.0001000...  output a
.a

CSE 521 - Arithmetic Coding - Spring 2003 28
```

### Decoding (3)

- Assume the length is known to be 3.
- 0001 which converts to the tag .0001000...

```
.0001000...  output b
.a

CSE 521 - Arithmetic Coding - Spring 2003 29
```

### Arithmetic Decoding Algorithm

- \( P(a_1), P(a_2), \ldots, P(a_m) \)
- \( C(a) = P(a_1) + P(a_2) + \ldots + P(a_i) \)
- Decode \( b_1b_2\ldots b_m \) number of symbols is \( n \).

1. Initialize \( L := 0 \) and \( R := 1 \);
2. For \( i = 1 \) to \( n \) do:
   1. \( W := R - L \);
   2. Find \( j \) such that \( L + W \cdot C(a_j) \leq t < L + W \cdot (C(a_j) + P(a_j)) \)
   3. Output \( a_j \);
   4. \( L := L + W \cdot C(a_j) \);
   5. \( R := L + W \cdot P(a_j) \);
Decoding Example

• P(a) = 1/4, P(b) = 1/2, P(c) = 1/4
• C(a) = 0, C(b) = 1/4, C(c) = 3/4

00101

tag = .00101000... = 5/32

<table>
<thead>
<tr>
<th>W</th>
<th>L</th>
<th>R</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>1/4</td>
<td>1/16</td>
<td>3/16</td>
<td>b</td>
</tr>
<tr>
<td>1/8</td>
<td>5/32</td>
<td>6/32</td>
<td>c</td>
</tr>
<tr>
<td>1/32</td>
<td>5/32</td>
<td>21/128</td>
<td>a</td>
</tr>
</tbody>
</table>

Decoding Issues

• There are two ways for the decoder to know when to stop decoding.
  1. Transmit the length of the string
  2. Transmit a unique end of string symbol

More Issues

• Avoiding real arithmetic and scaling
• Context
• Adaptive
• Comparison with Huffman coding

Scaling

• Scaling:
  – By scaling we can keep L and R in a reasonable range of values so that W = R – L does not underflow.
  – The code can be produced progressively, not at the end.
  – Complicates decoding some.

Scaling Principle

Lower half
  If [L,R) is contained in [0,.5) then
  L := 2L, R := 2R
  output 0, followed by C 1's
  C := 0.

Upper half
  If [L,R) is contained in [.5,1) then
  L := 2L - 1, R := 2R - 1
  output 1, followed by C 0's
  C := 0.

Middle Half
  If [L,R) is contained in [.25,.75) then
  L := 2L - .5, R := 2R - .5
  C := C + 1.

Example

• baa
Example

- baa

\[
\begin{align*}
C & = 0 \\
L & = 1/3 \quad R = 3/3 \\
L & = 3/9 \quad R = 5/9 \\
1/3 & \quad 2/3
\end{align*}
\]

Scale middle half

Example

- ba

\[
\begin{align*}
C & = 1 \\
L & = 3/9 \quad R = 5/9 \\
L & = 3/18 \quad R = 11/18 \\
1/3 & \quad 2/3
\end{align*}
\]

C = 1

Example

- bag 01

\[
\begin{align*}
C & = 0 \\
L & = 9/54 \quad R = 17/54 \\
L & = 18/54 \quad R = 34/54 \\
2/3 & \quad 1
\end{align*}
\]

Scale lower half

Example

- baa 01

In end L < 1/2 < R, choose tag to be 1/2

\[
\begin{align*}
C & = 0 \\
L & = 9/54 \quad R = 17/54 \\
L & = 18/54 \quad R = 34/54 \\
2/3 & \quad 1
\end{align*}
\]

Integer Implementation

- m bit integers
  - Represent 0 with 000...0 (m times)
  - Represent 1 with 111...1 (m times)

- Probabilities represented by frequencies
  - \( n_i \) is the number of times that symbol \( a_i \) occurs
  - \[ C_i = n_1 + n_2 + \ldots + n_{i-1} \]
  - \[ N = n_1 + n_2 + \ldots + n_m \]
  - \[ W = R/L + 1 \]

Coding the i-th symbol using integer calculations. Must use scaling!
- **Context**
  - Consider 1 symbol context.
  - Example: 3 contexts.

```
+---+---+---+
| a | b | c |
+---+---+---+
| .4 | .2 | .4 |
+---+---+---+
| .1 | .8 | .1 |
+---+---+---+
| .25| .25| .25|
+---+---+---+
```

- **Example with Scaling**

```
+---+---+---+
| a | b | c |
+---+---+---+
| .4 | .2 | .4 |
+---+---+---+
| .1 | .8 | .1 |
+---+---+---+
| .25| .25| .25|
+---+---+---+
```

- **Arithmetic Coding with Context**
  - Maintain the probabilities for each context.
  - For the first symbol use the equal probability model.
  - For each successive symbol use the model for the previous symbol.

- **Adaptation**
  - Simple solution – Equally Probable Model.
    - Initially all symbols have frequency 1.
    - After symbol x is coded, increment its frequency by 1.
    - Use the new model for coding the next symbol.
  - Example in alphabet a,b,c,d

```
a 1 2 3 4 5 6 7 8 9 10
b 1 1 1 1 1 1 1 1 1 1
<esc> 1 1 1 1 1 1 1 1 1 1
```

- **Zero Frequency Problem**
  - How do we weight symbols that have not occurred yet.
    - Equal weights? Not so good with many symbols
    - Escape symbol, but what should its weight be?
    - When a new symbol is encountered send the <esc>, followed by the symbol in the equally probable model. (Both encoded arithmetically.)

```
a  a  b  a  c
a 0 1 2 3 4 5 6 7 8 9 10
b 1 1 1 1 1 1 1 1 1 1 1
<esc> 1 1 1 1 1 1 1 1 1 1 1
```

- **PPM**
  - Prediction with Partial Matching
    - Cleary and Witten (1984)
  - State of the art arithmetic coder
    - Arbitrary order context
    - The context chosen is one that does a good prediction given the past
    - Adaptive
  - Example
    - Context “the” does not predict the next symbol “a” well. Move to the context “he” which does.
Arithmetic vs. Huffman

• Both compress very well. For m symbol grouping.
  – Huffman is within 1/m of entropy.
  – Arithmetic is within 2/m of entropy.
• Context
  – Huffman needs a tree for every context.
  – Arithmetic needs a small table of frequencies for every context.
• Adaptation
  – Huffman has an elaborate adaptive algorithm
  – Arithmetic has a simple adaptive mechanism.
• Bottom Line – Arithmetic is more flexible than Huffman.

Applications of Arithmetic Coding

• JPEG 2000
  – Image compression
  – Wavelet transform
  – Bit-planes of the transformed image is adaptively arithmetic coded.
  – Contexts relate to structure of wavelet coefficients
• JBIG
  – Binary image compression
  – Context is about 10 nearby pixels already coded.