CSE 490 GZ
Introduction to Data Compression
Winter 2006

Arithmetic Coding:
Scaling, Context, Adaptation

Scaling

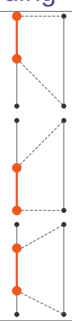
- Scaling:
  - By scaling we can keep L and R in a reasonable range of values so that W = R – L does not underflow.
  - The code can be produced progressively, not at the end.
  - Complicates decoding some.

Scaling during Encoding

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

C = 0

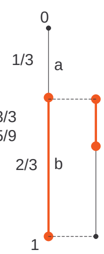
L = 1/3 R = 3/3

Example

- baa

C = 0

L = 1/3 R = 3/3
L = 3/9 R = 5/9



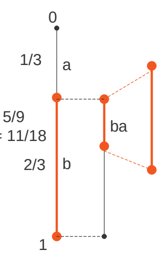
Scale middle half

Example

- baa

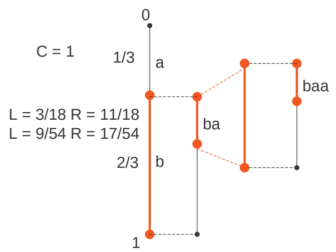
C = 1

L = 3/9 R = 5/9
L = 3/18 R = 11/18



Example

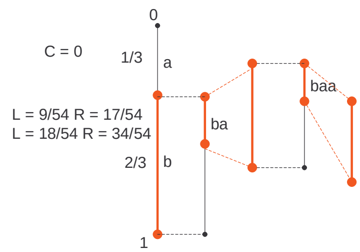
- baa



Scale lower half

Example

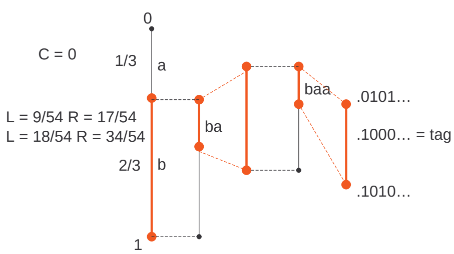
- baa 01



Example

- baa 011

In end L < ½ < R, choose tag to be 1/2



Exercise

Model: a: 1/4; b: 3/4
Encode: bba

Decoding

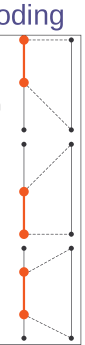
- The decoder behaves just like the encoder except that C does not need to be maintained.
- Instead, the input stream is consumed during when scaling.

Scaling during Decoding

Lower half
If [L,R) is contained in [0,.5) then
L := 2L; R := 2R
consume 0 from the encoded stream

Upper half
If [L,R) is contained in [.5,1) then
L := 2L –1, R := 2R - 1
consume 1 from the encoded stream

Middle half
If [L,R) is contained in [.25,.75) then
L := 2L –.5, R := 2R -.5
Replace 01 with 0 on stream
Replace 10 with 1 on stream



Scaling Math for the Tag

- Lower Half
  - $.0b_1b_2\ldots \times 10 = .b_1b_2$
- Upper Half
  - $.1b_1b_2\ldots \times 10 - 1 = .b_1b_2$
- Middle Half
  - $.01b_2b_3\ldots \times 10 - .1 = .0b_2b_3$
  - $.10b_2b_3\ldots \times 10 - .1 = .1b_2b_3$

Exercise

Model: a: 1/4; b: 3/4
Decode: 001 to 3 symbols

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
  - $C_i = n_1 + n_2 + \ldots + n_{i-1}$
  - $N = n_1 + n_2 + \ldots + n_m$

$$W := R - L + 1$$
$$L' := L + \left\lfloor \frac{W \cdot C_i}{N} \right\rfloor$$
$$R := L + \left\lfloor \frac{W \cdot C_{i+1}}{N} \right\rfloor - 1$$
$$L := L'$$

Coding the i-th symbol using integer calculations.
Must use scaling!

Context

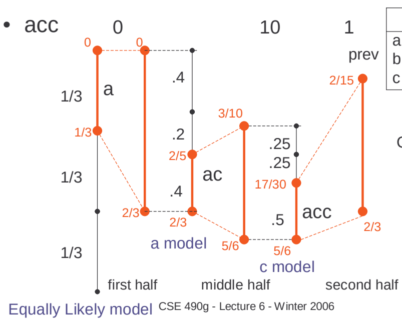
- Consider 1 symbol context.
- Example: 3 contexts.

prev / next

	a	b	c
a	.4	.2	.4
b	.1	.8	.1
c	.25	.25	.5

Example with Scaling

- acc



prev / next

	a	b	c
a	.4	.2	.4
b	.1	.8	.1
c	.25	.25	.5

Code = 0101

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	a	b	a	a	c
a	1	2	3	3	4	5	5
b	1	1	1	2	2	2	2
c	1	1	1	1	1	1	2
d	1	1	1	1	1	1	1

After aabaac is encoded
The probability model is
a 5/10 b 2/10
c 2/10 d 1/10

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	a	c
a	0	1	2	2	3	4	4
b	0	0	0	1	1	1	1
c	0	0	0	0	0	0	1
d	0	0	0	0	0	0	0
<esc>	1	1	1	1	1	1	1

After aabaac is encoded
The probability model is
a 4/7 b 1/7
c 1/7 d 0
<esc> 1/7

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.