Multimedia Information Retrieval and Technology

Lecture 20
Compression Algorithms III

By : Laura Liu

Room: EE314

Tel. no. 7756

Xi'an Jiaotong-Liverpool University
西交利物浦大学

1. Introduction

2. Variable-Length Coding (VLC)

- **Shannon-Fano Algorithm**
- **Huffman Coding Algorithm**
- **Adaptive Huffman Coding Algorithm**

3. Basics of Information Theory

4. LZW Compression

5. Arithmetic Coding

Arithmetic Coding

Arithmetic coding is a more modern coding method that usually outperforms Huffman coding.

- Huffman coding assigns each symbol a codeword which has an integral bit length.
- Arithmetic coding can treat the whole message as one unit.
- A message is represented by a half-open interval [*a, b*), where a and b are real numbers between 0 and 1.

Arithmetic Coding

Initially, the interval is [0,1).

When the message becomes longer, the length of the interval shortens, and the number of bits needed to represent the interval increases.

Example: encode symbols CAEE$

Suppose the alphabet is *[A, B,* C, *D, E, F,* $], in which $ is a special symbol used to terminate the message.

Probability distribution is as shown in Figure 7.8(a).

Symbol	Probability	Range
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3, 0.5)
D	0.05	[0.5, 0.55)
E	0.3	[0.55, 0.85)
F	0.05	[0.85, 0.9)
$	0.1	[0.9, 1.0)

(a)

ALGORITHM 7.5 Arithmetic Coding Encoder

```
BEGIN
    low = 0.0;    high = 1.0;    range = 1.0;

    while (symbol != terminator)
        {
          get (symbol);
          low =  low + range * Range_low(symbol);
          high = low + range * Range_high(symbol);
          range = high - low;
        }

    output a code so that low <= code < high;
END
```

Xi'an Jiaotong-Liverpool University
西交利物浦大學

Symbol	low	high	range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

(c) New *low*, *high*, and *range* generated.

Fig. 7.8 (cont'd): Arithmetic Coding: Encode Symbols "CAEE$"

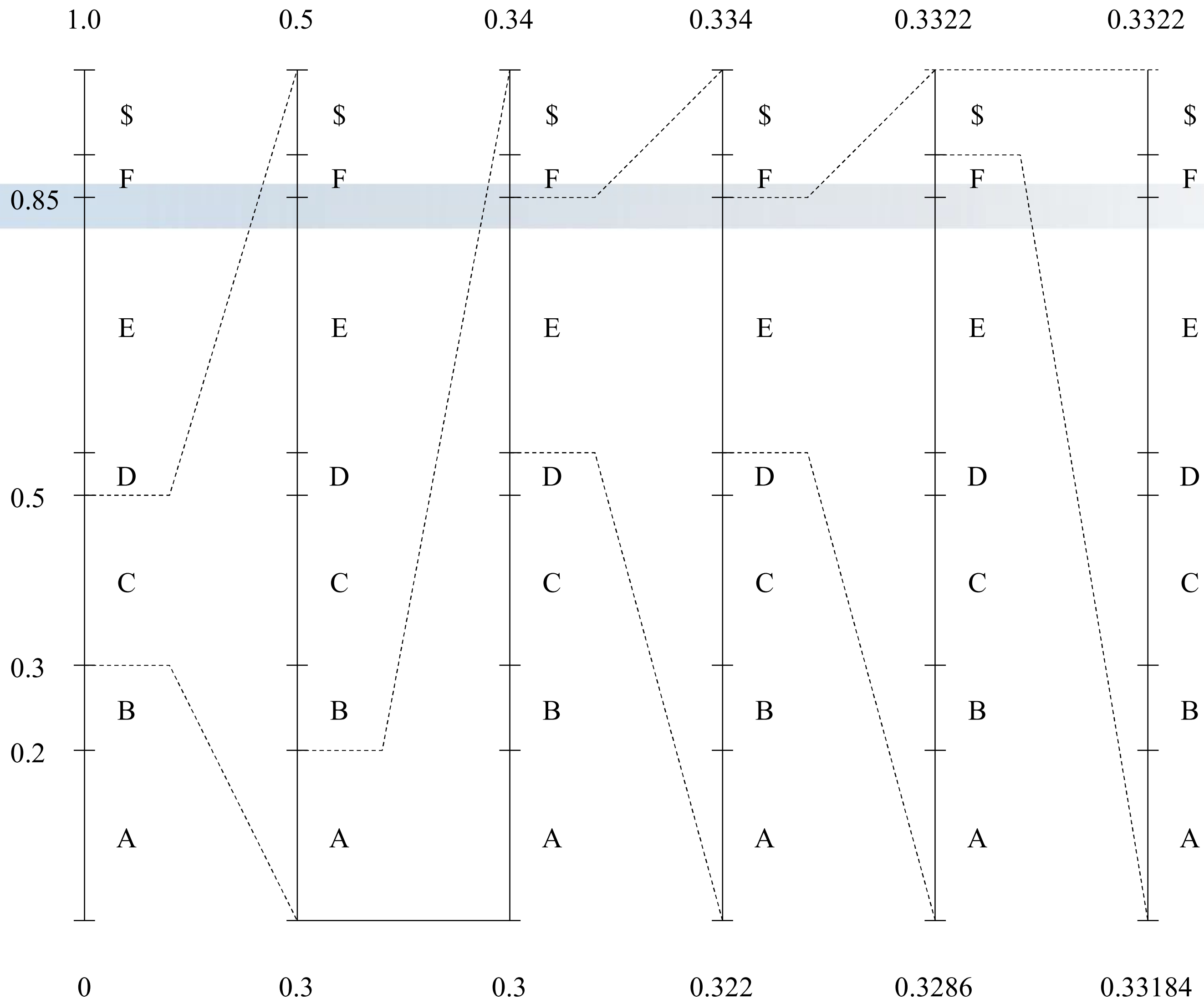

Fig. 7.8(b) Graphical display of shrinking ranges.

Arithmetic Coding

For clarity of illustration, the ever-shrinking ranges are enlarged in each step (indicated by dashed lines) in Figure 7.8(b).

After the second symbol A, *low, high,* and *range* are 0.30, 0.34, and 0.04.

The process repeats itself until after the terminating symbol $ is received. By then *low* and *high* are 0.33184 and 0.33220, respectively.

Arithmetic Coding

It is apparent that finally we have

$$range = P_C \times P_A \times P_E \times P_E \times P_\$ = 0.2 \times 0.2 \times 0.3 \times 0.3 \times 0.1 = 0.00036$$

The final step in encoding calls for generation of a number that falls within the range *[low, high).*

[0.33184, 0.33220)

Arithmetic Coding

The following algorithm will ensure that **the shortest binary codeword** is found if *low* and *high* are the two ends of the range and *low* < *high* .

Generating Codeword for Encoder

```
BEGIN
    code = 0;
    k = 1;
    While (value(code)<low)
        { assign 1 to the kth binary fraction bit;
          if (value(code) > high)
              Replace the kth bit by 0;
          k = k + 1;
        }
END
```

- The final step in Arithmetic encoding calls for the generation of a number that falls within the range $[low, high)$. The above algorithm will ensure that the shortest binary codeword is found.

Arithmetic Coding

When *low* = 0.33184, *high* = 0.3322. If we assign 1 to the first binary fraction bit, it would be 0.1 in binary, and its decimal *value(code)* = *value(0.1)* $= 2^{-1} = 0.5$ > *high.* Hence, we assign 0 to the first bit. Since *value(0.0)*= 0 < *low,* the while loop continues.

Assigning 1 to the second bit makes a binary *code* 0.01 and *value(0.01)*$=2^{-2}= 0.25$, which is less than *high,* so it is accepted.

Since it is still true that *value(0.01)* < *low,* the iteration continues.

Arithmetic Coding

Eventually, the binary codeword generated is 0.01010101, which is

$$2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} = 0.33203125$$

8 bits in total.

1. Introduction
2. Variable-Length Coding (VLC)
   - **Shannon-Fano Algorithm**
   - **Huffman Coding Algorithm**
   - **Adaptive Huffman Coding Algorithm**
3. Basics of Information Theory
4. LZW Compression
5. Arithmetic Coding

Xi'an Jiaotong-Liverpool University
西交利物浦大學

ALGORITHM 7.6 ARITHMETIC CODING DECODER

```
BEGIN
   get binary code and convert to decimal value = value(code);
   Do
        {
          find a symbol s so that
               Range_low(s) <= value < Range_high(s);
          output s;
          low = Rang_low(s);
          high = Range_high(s);
          range = high - low;
          value = [value - low] / range;
        }
   Until symbol s is a terminator
END
```

Arithmetic Decoding

Table 7.5 illustrates the decoding process for the above example.

Initially, *value* =0.33203125. Since this value falls in *Range(C),* the first output symbol is C.

This yields *value* = [0.33203125 - 0.3]/0.2 = 0.16015625, which in turn determines that the second symbol is A.

Eventually, *value* is 0.953125, which falls in the range [0.9, 1.0) of the terminator $.

Table 7.5 Arithmetic coding: decode symbols "CAEE$"

value	Output Symbol	low	high	range
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	$	0.9	1.0	0.1

Exercise I

Suppose the alphabet is *[A, B,* C], and the known probability distribution is P_A = 0.5, P_B = 0.4, P_C = 0.1. For simplicity, let's also assume that both encoder and decoder know that the length of the messages is always 3, so there is no need for a terminator.

i. How many bits are needed to encode the message BBB by Huffman coding?

ii. How many bits are needed to encode the message BBB by arithmetic coding?

Properties of Arithmetic Coding

1) **Variable-length codes**
2) **Nonblock codes**: an entire sequence of source symbols is assigned a single arithmetic code word;