7. Entropy source coding and data compression

Coding Technology

Source coding and data compression

In any text, different characters typically have different frequencies. Normal coding (without compression) means that all characters are coded using the same amount of bits.

Source coding and data compression

In any text, different characters typically have different frequencies. Normal coding (without compression) means that all characters are coded using the same amount of bits.

We will allow different characters to have varying length codewords in order to obtain a lower average codeword length.

Source coding and data compression

In any text, different characters typically have different frequencies. Normal coding (without compression) means that all characters are coded using the same amount of bits.

We will allow different characters to have varying length codewords in order to obtain a lower average codeword length.

A coding is *prefix-free* if none of the codewords is a prefix of another codeword. This property is necessary for decoding.

Source coding and data compression

In any text, different characters typically have different frequencies. Normal coding (without compression) means that all characters are coded using the same amount of bits.

We will allow different characters to have varying length codewords in order to obtain a lower average codeword length.

A coding is *prefix-free* if none of the codewords is a prefix of another codeword. This property is necessary for decoding.

We assume that the distribution (long-term frequency) of characters in the text is known: the probabilities of the characters are

$$p_1, \ldots, p_K,$$

where K is the size of the alphabet.

Source coding and data compression

If a coding assigns a codeword of length ℓ_k to character k, then the *average codelength* is

$$L = \sum_{k=1}^{K} p_k \ell_k .$$

Source coding and data compression

If a coding assigns a codeword of length ℓ_k to character k, then the *average codelength* is

$$L = \sum_{k=1}^{K} p_k \ell_k .$$

The *entropy* of the text source is

$$H(X) = \sum_{k=1}^{K} p_k \log_2(1/p_k).$$

Theoretical lower bound: for any prefix-free coding,

$$L \geq H(X),$$

and the ratio $H(X)/L$ is called the *efficiency* of the code.

Shannon–Fano coding

For the Shannon–Fano coding, the codeword lengths are

$$\ell_k = \lceil \log_2(1/p_k) \rceil.$$

We construct a binary tree where the depths of the leaves are $\ell_1, \ldots, \ell_K$, and the codewords will be based on the route from the root to the leaves.

Shannon–Fano coding

For the Shannon–Fano coding, the codeword lengths are

$$\ell_k = \lceil \log_2(1/p_k) \rceil.$$

We construct a binary tree where the depths of the leaves are $\ell_1, \dots, \ell_K$, and the codewords will be based on the route from the root to the leaves.

Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

Shannon–Fano coding

For the Shannon–Fano coding, the codeword lengths are

$$\ell_k = \lceil \log_2(1/p_k) \rceil.$$

We construct a binary tree where the depths of the leaves are $\ell_1, \ldots, \ell_K$, and the codewords will be based on the route from the root to the leaves.

Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

$$\ell_1 = \lceil \log_2(1/0.37) \rceil = 2, \qquad \ell_2 = \lceil \log_2(1/0.27) \rceil = 2,$$
$$\ell_3 = \lceil \log_2(1/0.24) \rceil = 3, \qquad \ell_4 = \lceil \log_2(1/0.12) \rceil = 4.$$

Shannon–Fano coding

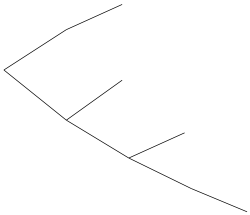
For the Shannon–Fano coding, the codeword lengths are

$$\ell_k = \lceil \log_2(1/p_k) \rceil.$$

We construct a binary tree where the depths of the leaves are $\ell_1, \ldots, \ell_K$, and the codewords will be based on the route from the root to the leaves.

Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

$$\ell_1 = \lceil \log_2(1/0.37) \rceil = 2, \qquad \ell_2 = \lceil \log_2(1/0.27) \rceil = 2,$$
$$\ell_3 = \lceil \log_2(1/0.24) \rceil = 3, \qquad \ell_4 = \lceil \log_2(1/0.12) \rceil = 4.$$



Shannon–Fano coding

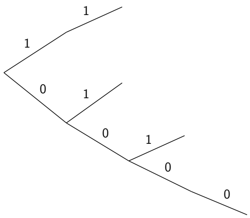
For the Shannon–Fano coding, the codeword lengths are

$$\ell_k = \lceil \log_2(1/p_k) \rceil.$$

We construct a binary tree where the depths of the leaves are $\ell_1, \ldots, \ell_K$, and the codewords will be based on the route from the root to the leaves.

Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

$$\ell_1 = \lceil \log_2(1/0.37) \rceil = 2, \qquad \ell_2 = \lceil \log_2(1/0.27) \rceil = 2,$$
$$\ell_3 = \lceil \log_2(1/0.24) \rceil = 3, \qquad \ell_4 = \lceil \log_2(1/0.12) \rceil = 4.$$



Shannon–Fano coding

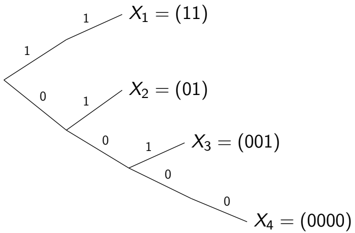
For the Shannon–Fano coding, the codeword lengths are

$$\ell_k = \lceil \log_2(1/p_k) \rceil.$$

We construct a binary tree where the depths of the leaves are $\ell_1, \ldots, \ell_K$, and the codewords will be based on the route from the root to the leaves.

Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

$$\ell_1 = \lceil \log_2(1/0.37) \rceil = 2, \qquad \ell_2 = \lceil \log_2(1/0.27) \rceil = 2,$$
$$\ell_3 = \lceil \log_2(1/0.24) \rceil = 3, \qquad \ell_4 = \lceil \log_2(1/0.12) \rceil = 4.$$



Shannon–Fano coding

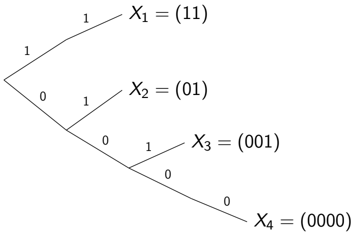
For the Shannon–Fano coding, the codeword lengths are

$$\ell_k = \lceil \log_2(1/p_k) \rceil.$$

We construct a binary tree where the depths of the leaves are $\ell_1, \ldots, \ell_K$, and the codewords will be based on the route from the root to the leaves.

Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

$$\ell_1 = \lceil \log_2(1/0.37) \rceil = 2, \qquad \ell_2 = \lceil \log_2(1/0.27) \rceil = 2,$$
$$\ell_3 = \lceil \log_2(1/0.24) \rceil = 3, \qquad \ell_4 = \lceil \log_2(1/0.12) \rceil = 4.$$



Symbol	Codeword
X_1	11
X_2	01
X_3	001
X_4	0000

Problem 1

Encode the following distribution using Shannon–Fano coding.

$$p_1 = 0.49, \quad p_2 = 0.14, \quad p_3 = 0.14, \quad p_4 = 0.07, \quad p_5 = 0.07,$$
$$p_6 = 0.04, \quad p_7 = 0.02, \quad p_8 = 0.02, \quad p_9 = 0.01$$

Problem 1

Encode the following distribution using Shannon–Fano coding.

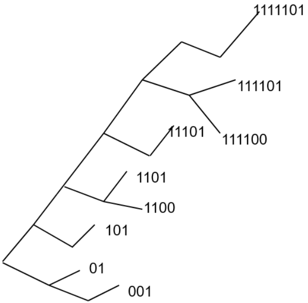
$$p_1 = 0.49, \quad p_2 = 0.14, \quad p_3 = 0.14, \quad p_4 = 0.07, \quad p_5 = 0.07,$$
$$p_6 = 0.04, \quad p_7 = 0.02, \quad p_8 = 0.02, \quad p_9 = 0.01$$

Solution. Codeword lengths: $\ell_i = \lceil \log_2 1/p_i \rceil$, so

$$\begin{aligned}
\ell_1 &= \lceil \log_2 1/p_1 \rceil = \lceil 1.029 \rceil = 2, \\
\ell_2 &= \lceil \log_2 1/p_2 \rceil = \lceil 2.836 \rceil = 3, \\
\ell_3 &= \lceil \log_2 1/p_3 \rceil = \lceil 2.836 \rceil = 3, \\
\ell_4 &= \ell_5 = 4, \quad \ell_6 = 5, \quad \ell_7 = \ell_8 = 6, \quad \ell_9 = 7.
\end{aligned}$$

(Instead of $\log_2$, the notation ld is also in use.)

Problem 1



Symbol	Codeword
X_1	01
X_2	001
X_3	101
X_4	1100
X_5	1101
X_6	11101
X_7	111100
X_8	111101
X_9	1111101

Side note: prefix-free code $\Leftrightarrow$ no codewords on inner nodes.

Problem 2

Conduct performance analysis for the previous code.

Problem 2

Conduct performance analysis for the previous code.

Solution. The entropy of the original distribution is

$$H(X) = \sum_{i=1}^{9} p_i \log_2 \left(\frac{1}{p_i} \right) = 2.314,$$

and the average codeword length for the coding is

$$\begin{aligned} L = & 0.49 \cdot 2 + 0.28 \cdot 3 + 0.28 \cdot 3 + 0.14 \cdot 4 + \\ & 0.04 \cdot 5 + 0.04 \cdot 6 + 0.01 \cdot 7 = 2.89, \end{aligned}$$

so the efficiency of the coding is

$$\frac{H(X)}{L} \approx 0.8.$$

Huffman coding

Huffman coding builds the tree by adding the two smallest p_k probabilities in each step. After that, the coding works the same as for Shannon–Fano.

Huffman coding

Huffman coding builds the tree by adding the two smallest p_k probabilities in each step. After that, the coding works the same as for Shannon–Fano.

Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

$p_1 = 0.37$

$p_2 = 0.27$

$p_3 = 0.24$

$p_4 = 0.12$

Huffman coding

Huffman coding builds the tree by adding the two smallest p_k probabilities in each step. After that, the coding works the same as for Shannon–Fano.

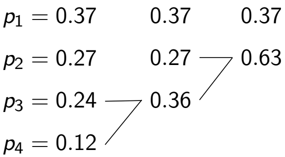
Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

$p_1 = 0.37$	0.37
$p_2 = 0.27$	0.27
$p_3 = 0.24$	0.36
$p_4 = 0.12$	

Huffman coding

Huffman coding builds the tree by adding the two smallest p_k probabilities in each step. After that, the coding works the same as for Shannon–Fano.

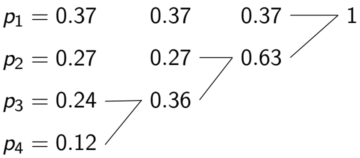
Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$



Huffman coding

Huffman coding builds the tree by adding the two smallest p_k probabilities in each step. After that, the coding works the same as for Shannon–Fano.

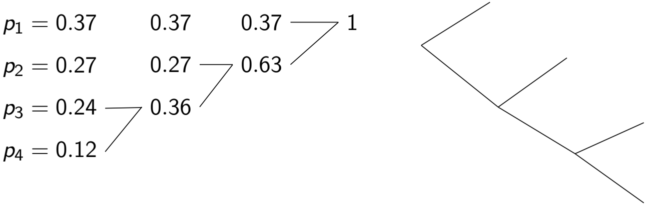
Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$



Huffman coding

Huffman coding builds the tree by adding the two smallest p_k probabilities in each step. After that, the coding works the same as for Shannon–Fano.

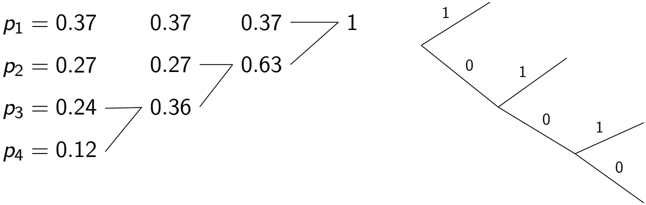
Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$



Huffman coding

Huffman coding builds the tree by adding the two smallest p_k probabilities in each step. After that, the coding works the same as for Shannon–Fano.

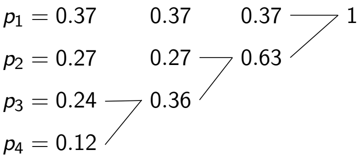
Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$

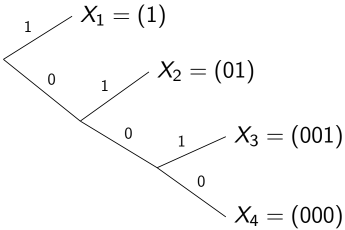


Huffman coding

Huffman coding builds the tree by adding the two smallest p_k probabilities in each step. After that, the coding works the same as for Shannon–Fano.

Example. $p_1 = 0.37, \quad p_2 = 0.27, \quad p_3 = 0.24, \quad p_4 = 0.12.$





Problem 3

Encode the source of problem 1 by Huffman coding.

Problem 3

Encode the source of problem 1 by Huffman coding.

Solution. First the state graph is constructed.

$p_1 = 0.49$

$p_2 = 0.14$

$p_3 = 0.14$

$p_4 = 0.07$

$p_5 = 0.07$

$p_6 = 0.04$

$p_7 = 0.02$

$p_8 = 0.02$

$p_9 = 0.01$

Problem 3

Encode the source of problem 1 by Huffman coding.

Solution. First the state graph is constructed.

$p_1 = 0.49$	0.49
$p_2 = 0.14$	0.14
$p_3 = 0.14$	0.14
$p_4 = 0.07$	0.07
$p_5 = 0.07$	0.07
$p_6 = 0.04$	0.04
$p_7 = 0.02$	0.02
$p_8 = 0.02$	0.03
$p_9 = 0.01$	

Problem 3

Encode the source of problem 1 by Huffman coding.

Solution. First the state graph is constructed.

$p_1 = 0.49$	0.49	0.49
$p_2 = 0.14$	0.14	0.14
$p_3 = 0.14$	0.14	0.14
$p_4 = 0.07$	0.07	0.07
$p_5 = 0.07$	0.07	0.07
$p_6 = 0.04$	0.04	0.04
$p_7 = 0.02$	0.02	0.05
$p_8 = 0.02$	0.03	
$p_9 = 0.01$		

Problem 3

Encode the source of problem 1 by Huffman coding.

Solution. First the state graph is constructed.

$p_1 = 0.49$	0.49	0.49	0.49
$p_2 = 0.14$	0.14	0.14	0.14
$p_3 = 0.14$	0.14	0.14	0.14
$p_4 = 0.07$	0.07	0.07	0.07
$p_5 = 0.07$	0.07	0.07	0.07
$p_6 = 0.04$	0.04	0.04	0.09
$p_7 = 0.02$	0.02	0.05	
$p_8 = 0.02$	0.03		
$p_9 = 0.01$			

Problem 3

Encode the source of problem 1 by Huffman coding.

Solution. First the state graph is constructed.

$p_1 = 0.49$	0.49	0.49	0.49	0.49
$p_2 = 0.14$	0.14	0.14	0.14	0.14
$p_3 = 0.14$	0.14	0.14	0.14	0.14
$p_4 = 0.07$	0.07	0.07	0.07	0.14
$p_5 = 0.07$	0.07	0.07	0.07	0.09
$p_6 = 0.04$	0.04	0.04	0.09	
$p_7 = 0.02$	0.02	0.05		
$p_8 = 0.02$	0.03			
$p_9 = 0.01$				

Problem 3

Encode the source of problem 1 by Huffman coding.

Solution. First the state graph is constructed.

$p_1 = 0.49$	0.49	0.49	0.49	0.49	0.49
$p_2 = 0.14$	0.14	0.14	0.14	0.14	0.14
$p_3 = 0.14$	0.14	0.14	0.14	0.14	0.14
$p_4 = 0.07$	0.07	0.07	0.07	0.14	0.23
$p_5 = 0.07$	0.07	0.07	0.07	0.09	
$p_6 = 0.04$	0.04	0.04	0.09		
$p_7 = 0.02$	0.02	0.05			
$p_8 = 0.02$	0.03				
$p_9 = 0.01$					

Problem 3

Encode the source of problem 1 by Huffman coding.

Solution. First the state graph is constructed.

$p_1 = 0.49$	0.49	0.49	0.49	0.49	0.49	0.49
$p_2 = 0.14$	0.14	0.14	0.14	0.14	0.14	0.28
$p_3 = 0.14$	0.14	0.14	0.14	0.14	0.14	0.23
$p_4 = 0.07$	0.07	0.07	0.07	0.14	0.23	
$p_5 = 0.07$	0.07	0.07	0.07	0.09		
$p_6 = 0.04$	0.04	0.04	0.09			
$p_7 = 0.02$	0.02	0.05				
$p_8 = 0.02$	0.03					
$p_9 = 0.01$						

Problem 3

Encode the source of problem 1 by Huffman coding.

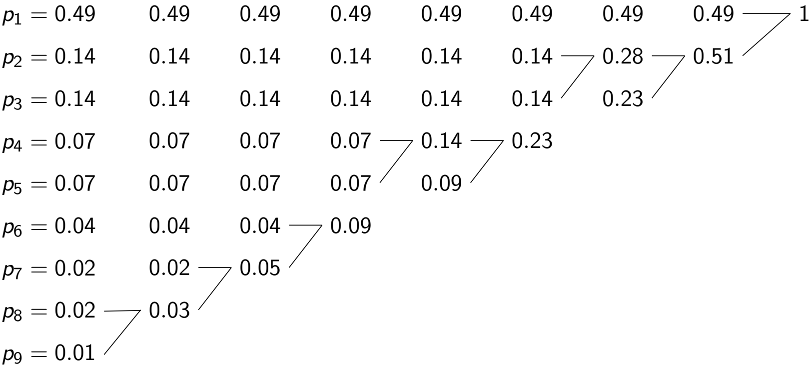
Solution. First the state graph is constructed.

$p_1 = 0.49$	0.49	0.49	0.49	0.49	0.49	0.49	0.49
$p_2 = 0.14$	0.14	0.14	0.14	0.14	0.14	0.28	0.51
$p_3 = 0.14$	0.14	0.14	0.14	0.14	0.14	0.23	
$p_4 = 0.07$	0.07	0.07	0.07	0.14	0.23		
$p_5 = 0.07$	0.07	0.07	0.07	0.09			
$p_6 = 0.04$	0.04	0.04	0.09				
$p_7 = 0.02$	0.02	0.05					
$p_8 = 0.02$	0.03						
$p_9 = 0.01$							

Problem 3

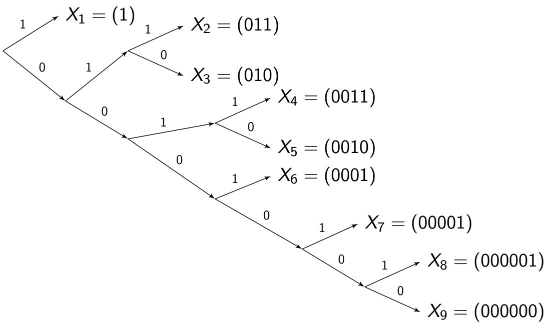
Encode the source of problem 1 by Huffman coding.

Solution. First the state graph is constructed.



Problem 3

Then the code tree and coding LUT can be obtained:



Symbol	Codeword
X_1	1
X_2	011
X_3	010
X_4	0010
X_5	0011
X_6	0001
X_7	00001
X_8	000001
X_9	000000

Problem 4

Compare the performance of the Shannon-Fano coding and the Huffman coding for the previous source for sampling frequency $f_s = 160$ MHz.

Problem 4

Compare the performance of the Shannon-Fano coding and the Huffman coding for the previous source for sampling frequency $f_s = 160$ MHz.

Solution. We first compute the average codelength for both HUFF and SF coding.

$$\begin{aligned} L^{HUFF} &= 0.49 \cdot 1 + 0.14 \cdot 3 + 0.14 \cdot 3 + 0.07 \cdot 4 + 0.07 \cdot 4 + \\ &\quad + 0.04 \cdot 4 + 0.02 \cdot 5 + 0.02 \cdot 6 + 0.01 \cdot 6 = 2.33 \\ L^{SF} &= 0.49 \cdot 2 + 0.28 \cdot 3 + 0.14 \cdot 4 + 0.04 \cdot 5 + 0.04 \cdot 6 + \\ &\quad + 0.01 \cdot 7 = 2.89 \end{aligned}$$

Problem 4

Compare the performance of the Shannon-Fano coding and the Huffman coding for the previous source for sampling frequency $f_s = 160$ MHz.

Solution. We first compute the average codelength for both HUFF and SF coding.

$$\begin{aligned} L^{HUFF} &= 0.49 \cdot 1 + 0.14 \cdot 3 + 0.14 \cdot 3 + 0.07 \cdot 4 + 0.07 \cdot 4 + \\ &+ 0.04 \cdot 4 + 0.02 \cdot 5 + 0.02 \cdot 6 + 0.01 \cdot 6 = 2.33 \\ L^{SF} &= 0.49 \cdot 2 + 0.28 \cdot 3 + 0.14 \cdot 4 + 0.04 \cdot 5 + 0.04 \cdot 6 + \\ &+ 0.01 \cdot 7 = 2.89 \end{aligned}$$

At $f_s = 160$ MHz, the rates are

$$R_{HUFF} = 372.8 Mbps, \qquad R_{SF} = 462 Mbps.$$

Side note: 9 source symbols $\rightarrow$ without compression, 4 bits are required, and the rate is $R = 640 Mbps$.

Problem 5

We have a source with the following distribution and code table:

Source symbol	Probability	Codeword
X_1	0.4	0
X_2	0.2	10
X_3	0.2	110
X_4	0.2	1111

(a) Is this a prefix-free code?

(b) What is the average codelength?

(c) How far is the average codelength from the theoretical lower bound of compressibility?

(d) Is this an optimal code?

Problem 5

Solution.

(a) Yes, the code is prefix-free.

Problem 5

Solution.

(a) Yes, the code is prefix-free.

(b) $L = \sum_{i=1}^{4} p_i \ell_i = 0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.2 \cdot 4 = 2.2.$

Problem 5

Solution.

(a) Yes, the code is prefix-free.

(b) $L = \sum_{i=1}^{4} p_i \ell_i = 0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.2 \cdot 4 = 2.2.$

(c)

$$H(X) = \sum_{i=1}^{4} p_i \log_2 \left(\frac{1}{p_i} \right) = 0.4 \cdot 1.31 + 3 \cdot 0.2 \cdot 2.322 = 1.922$$

$$L - H(X) = 0.278$$

Problem 5

Solution.

(a) Yes, the code is prefix-free.

(b) $L = \sum_{i=1}^{4} p_i \ell_i = 0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.2 \cdot 4 = 2.2.$

(c)

$$H(X) = \sum_{i=1}^{4} p_i \log_2 \left(\frac{1}{p_i} \right) = 0.4 \cdot 1.31 + 3 \cdot 0.2 \cdot 2.322 = 1.922$$

$$L - H(X) = 0.278$$

(d) No, for X_4 the codeword 111 is sufficient instead of 1111. (The resulting code has the same codelengths as Huffman-coding, so it is optimal.)

Problem 6

Consider the source from Problem 1:

$$p_1 = 0.49, \quad p_2 = 0.14, \quad p_3 = 0.14, \quad p_4 = 0.07, \quad p_5 = 0.07,$$
$$p_6 = 0.04, \quad p_7 = 0.02, \quad p_8 = 0.02, \quad p_9 = 0.01.$$

(a) Compress the source using Shannon-Fano-Elias coding.

(b) Compute the average codelength.

(c) Compare the performance of this code with Shannon-Fano coding and Huffman coding for the same source for sampling frequency $f_s = 160$ MHz.

Problem 6

Solution.

(a)

i	p_i	$F(i)$	$\bar{F}(i)$	binary	ℓ_i	codeword
1	0.49	0	0.245	0.0011111010...	3	001
2	0.14	0.49	0.56	0.1000111101...	4	1000
3	0.14	0.63	0.7	0.1011001100...	4	1011
4	0.07	0.77	0.805	0.1100111000...	5	11001
5	0.07	0.84	0.875	0.1110000000...	5	11100
6	0.04	0.91	0.93	0.1110111000...	6	111011
7	0.02	0.95	0.96	0.1111010111...	7	1111010
8	0.02	0.97	0.98	0.1111101011...	7	1111101
9	0.01	0.99	0.995	0.1111111010...	8	11111110

$$F(i) = \sum_{j=0}^{i-1} p_i, \quad \bar{F}(i) = F(i) + p_i/2, \quad \ell_i = \lceil \log_2 (1/p_i) \rceil + 1$$

Problem 6

(b) Average codelength is

$$L^{SFE} = 0.49 \cdot 3 + 0.14 \cdot 4 + 0.14 \cdot 4 + 0.07 \cdot 5 + 0.07 \cdot 5 + \\ + 0.04 \cdot 6 + 0.02 \cdot 7 + 0.02 \cdot 7 + 0.01 \cdot 8 = 3.89.$$

Problem 6

(b) Average codelength is

$$L^{SFE} = 0.49 \cdot 3 + 0.14 \cdot 4 + 0.14 \cdot 4 + 0.07 \cdot 5 + 0.07 \cdot 5 + \\ + 0.04 \cdot 6 + 0.02 \cdot 7 + 0.02 \cdot 7 + 0.01 \cdot 8 = 3.89.$$

(c)

$$\begin{array}{ccc} L^{HUFF} = 2.33 & L^{SF} = 2.89 & L^{SFE} = 3.89 \\ \downarrow & \downarrow & \downarrow \\ R_{HUFF} = 372.8Mbps & R_{SF} = 462Mbps & R_{SFE} = 622Mbps \end{array}$$

Recall: without coding, $R = 640Mbps$.

Conclusion: small improvement in the average codelength L matters a lot in data speed!

Comparative analysis

performance ↑ $f_s = 160$ MHz alg. simplicity ↓

Code	Performance	Avg. length	Data speed	Complexity
Huffman	optimal L	2.33	372.8$Mbps$	search + tree
SF	$H(X) < L < H(X) + 1$	2.89	462.4$Mbps$	tree
SFE	$H(X) + 1 < L < H(X) + 2$	3.89	622.4$Mbps$	binary conv.

Arithmetic coding

Shannon–Fano–Elias coding was inefficient because the $\lfloor . \rfloor + 1$ function was applied to each character separately. Arithmetic coding (AC) is based on the same idea as SFE, but instead of coding characters separately, AC compresses the entire message at once.

Example. The alphabet is {A,B,C,D}, with

$$P(A) = 0.4, \quad P(B) = 0.3, \quad P(C) = 0.2, \quad P(D) = 0.1.$$

Arithmetic coding

Shannon–Fano–Elias coding was inefficient because the $\lfloor . \rfloor + 1$ function was applied to each character separately. Arithmetic coding (AC) is based on the same idea as SFE, but instead of coding characters separately, AC compresses the entire message at once.

Example. The alphabet is {A,B,C,D}, with

$$P(A) = 0.4, \quad P(B) = 0.3, \quad P(C) = 0.2, \quad P(D) = 0.1.$$

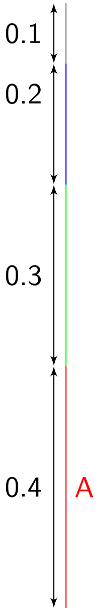
For AC, the compressed message will correspond to first a subinterval of $[0, 1)$, then a single point from $[0, 1)$.

Arithmetic coding – example

message: ABAC

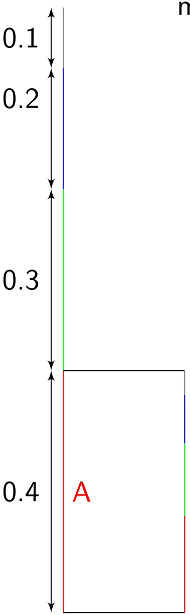
Arithmetic coding – example

message: ABAC



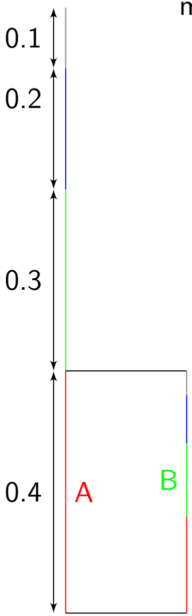
Arithmetic coding – example

message: ABAC

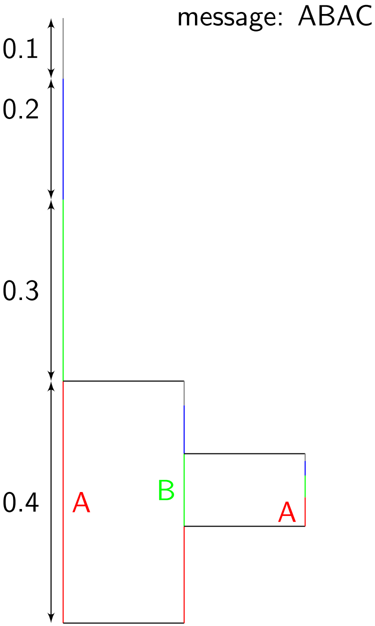


Arithmetic coding – example

message: ABAC

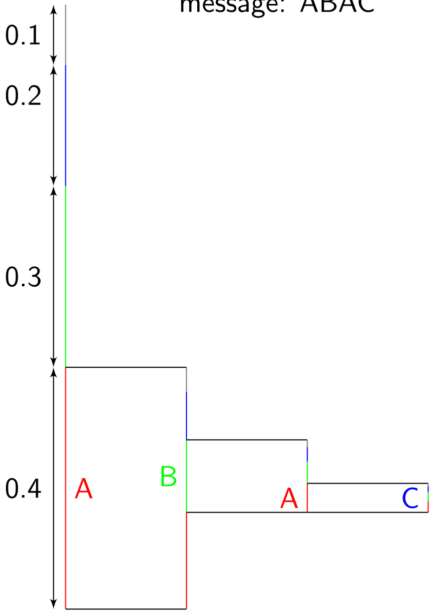


Arithmetic coding – example

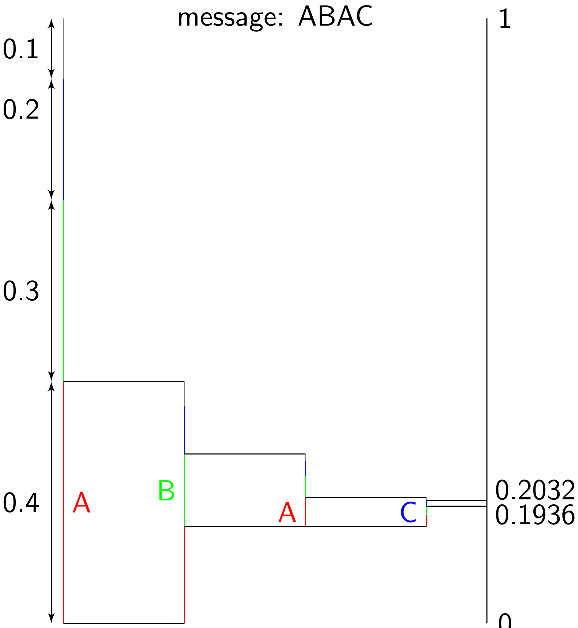


Arithmetic coding – example

message: ABAC



Arithmetic coding – example



Arithmetic coding – example

The interval corresponding to the message ABAC is $[0.1936, 0.2032]$.

We want to use the middle point of this interval (in binary form) as the compressed message:

$$0.1984 = 0.00110010110\ldots_{②}$$

Arithmetic coding – example

The interval corresponding to the message ABAC is $[0.1936, 0.2032]$.

We want to use the middle point of this interval (in binary form) as the compressed message:

$$0.1984 = 0.00110010110\ldots_{②}$$

The main question: how many bits of precision do we need so we can distinguish this interval from the other small intervals?

Arithmetic coding – example

The number of bits required is

$$\lceil -\log_2(P(A)P(B)P(A)P(C)) \rceil + 1 = 8,$$

because then the rounding error is smaller than $P(A)P(B)P(A)P(C)/2$, so even the rounded value will be inside the same interval:

$$\begin{aligned} 0.1936 &= 0.00110001100\ldots \\ 0.1984 &\approx 0.00110011 \\ 0.2032 &= 0.00110100000\ldots \end{aligned}$$

Arithmetic coding

AC is not character coding, so it can be better than Huffman coding. In fact, for long messages, the compression rate will asymptotically converge to the entropy lower bound: for a character sequence $C_1 \ldots C_n$,

$$\lim_{n\to\infty} \frac{1}{n}\left(\left\lceil -\log_2\left(\prod_{i=1}^{n} P(C_i)\right)\right\rceil + 1\right) = \lim_{n\to\infty} -\frac{1}{n}\log_2\left(\prod_{i=1}^{n} P(C_i)\right)$$

$$= \lim_{n\to\infty} -\frac{1}{n}\sum_{i=1}^{n} \log_2 P(C_i) = \sum_{k=1}^{K} p_k \log_2(1/p_k) = H(X)$$

due to the Law of Large Numbers.

Arithmetic coding

AC is not character coding, so it can be better than Huffman coding. In fact, for long messages, the compression rate will asymptotically converge to the entropy lower bound: for a character sequence $C_1 \ldots C_n$,

$$\lim_{n\to\infty} \frac{1}{n}\left(\left\lceil -\log_2\left(\prod_{i=1}^{n} P(C_i)\right)\right\rceil + 1\right) = \lim_{n\to\infty} -\frac{1}{n}\log_2\left(\prod_{i=1}^{n} P(C_i)\right)$$

$$= \lim_{n\to\infty} -\frac{1}{n}\sum_{i=1}^{n}\log_2 P(C_i) = \sum_{k=1}^{K} p_k \log_2(1/p_k) = H(X)$$

due to the Law of Large Numbers.

AC can be decompressed online: decoding can be started using the beginning of the compressed message, with more and more of the message decompressed as further sections of the compressed message are received.