

# Chapter 5

---

## VARIABLE-LENGTH CODING

---- *INFORMATION THEORY RESULTS (II)*

### 5.1 Some Fundamental Results

#### 5.1.1 Coding an Information Source

- Consider an information source, represented by a *source alphabet*  $S$ .

$$S = \{s_1, s_2, \dots, s_m\}, \quad (5.1)$$

where  $s_i$ 's are *source symbols*.

- Terms source symbol and information message.
  - Used interchangeably in the literature.

- We like to consider: an information message can be a source symbol, or a combination of source symbols.
- We denote *code alphabet* by  $A$  and

$$A = \{a_1, a_2, \dots, a_r\},$$

where  $a_j$ 's are *code symbols*.

- A *message code* is a sequence of code symbols that represents a given information message.
- In the simplest case, a message consists of only a source symbol.

Encoding is then a procedure to assign a *codeword* to the source symbol. Namely,

$$s_i \rightarrow A_i = (a_{i1}, a_{i2}, \dots, a_{ik}),$$

where the codeword  $A_i$  is a string of  $k$  code symbols assigned to the source symbol  $s_i$ .

- The term **message ensemble** is defined as the entire set of messages.

- A **code**, also known as an **ensemble code**, is defined as a mapping of all the possible sequences of symbols of  $S$  (message ensemble) into the sequences of symbols in  $A$ .
- In binary coding, the number of code symbols  $r$  is equal to 2, since there are only two code symbols available: the binary digits “0” and “1”.

### Example 5.1

- ✓ Consider an English article and ASCII code.
- ✓ In this context, the source alphabet consists of all the English letters in both lower and upper cases and all the punctuation marks.
- ✓ The code alphabet consists of the binary 1 and 0.
- ✓ There are a total of 128 7-bit binary codewords.
- ✓ From Table 5.1, we see that codeword assigned to capital letter A (a source symbol) is 1000001 (the codeword).

Table 5. 1 Seven-bit American standard code for information interchange (ASCII)

Bits				5	0	1	0	1	0	1	0	1
				6	0	0	1	1	0	0	1	1
1	2	3	4	7	0	0	0	0	1	1	1	1
0	0	0	0	NUL	DLE	SP	0	@	P	'	p	
1	0	0	0	SOH	DC1	!	1	A	Q	a	q	
0	1	0	0	STX	DC2	"	2	B	R	b	r	
1	1	0	0	ETX	DC3	#	3	C	S	c	s	
0	0	1	0	EOT	DC4	\$	4	D	T	d	t	
1	0	1	0	ENQ	NAK	%	5	E	U	e	u	
0	1	1	0	ACK	SYN	&	6	F	V	f	v	
1	1	1	0	BEL	ETB	'	7	G	W	g	w	
0	0	0	1	BS	CAN	(	8	H	X	h	x	
1	0	0	1	HT	EM	)	9	I	Y	i	y	
0	1	0	1	LF	SUB	*	:	J	Z	j	z	
1	1	0	1	VT	ESC	+	;	K	[	k	{	
0	0	1	1	FF	FS	,	<	L	\	l		
1	0	1	1	CR	GS	-	=	M	]	m	}	
0	1	1	1	SO	RS	.	>	N	^	n	~	
1	1	1	1	SI	US	/	?	O	—	o	DEL	

NUL	Null, or all zeros	DC1	Device control 1
SOH	Start of heading	DC2	Device control 2
STX	Start of text	DC3	Device control 3
ETX	End of text	DC4	Device control 4
EOT	End of transmission	NAK	Negative acknowledgment
ENQ	Enquiry	SYN	Synchronous idle
ACK	Acknowledge	ETB	End of transmission block
BEL	Bell, or alarm	CAN	Cancel
BS	Backspace	EM	End of medium
HT	Horizontal tabulation	SUB	Substitution
LF	Line feed	ESC	Escape
VT	Vertical tabulation	FS	File separator
FF	Form feed	GS	Group separator
CR	Carriage return	RS	Record separator
SO	Shift out	US	Unit separator
SI	Shift in	SP	Space
DLE	Data link escape	DEL	Delete

## Example 5.2

➤ Table 5.2 lists what is known as the (5,2) code.

It is a linear block code.

➤ In this example, the source alphabet consists of the four ( $2^2$ ) source symbols listed in the left column of the table: 00, 01, 10 and 11.

➤ The code alphabet consists of the binary 1 and 0.

➤ There are four codewords listed in the right column of the table.

➤ From the table, we see that the code assigns a 5-bit codeword to each source symbol.

Table 5.2 A (5,2) linear block code

Source Symbol	Codeword
$S_1 (00)$	00000
$S_2 (01)$	10100
$S_3 (10)$	01111
$S_4 (11)$	11011

## 5.1.2 Some Desired Characteristics

### 5.1.2.1 Block Code

- A code is said to be a block code if it maps each source symbol in  $S$  into a fixed codeword in  $A$ .
- Hence, the codes listed in the above two examples are block codes.

### 5.1.2.2 Uniquely Decodable Code

- A code is uniquely decodable if it can be unambiguously decoded.
- Obviously, a code has to be uniquely decodable if it is to be in use.

### Example 5.3

Table 5.3 A not uniquely decodable code

Source Symbol	Codeword
$S_1$	00
$S_2$	10
$S_3$	00
$S_4$	11

## Nonsingular Code

- A block code is nonsingular if all the codewords are distinct.

Table 5.4 A nonsingular code

Source Symbol	Codeword
$S_1$	1
$S_2$	11
$S_3$	00
$S_4$	01

### Example 5.4

- ✓ Table 5.4 gives a nonsingular code since all four codewords are distinct.
- ✓ If a code is not a nonsingular code, i.e., at least two codewords are identical, then the code is not uniquely decodable.
- ✓ Notice that, however, a nonsingular code does not guarantee unique decodability.
- ✓ The code shown in Table 5.4 is such an example in that it is nonsingular while it is not uniquely decodable (because once the binary string “11” is received, we do not know if the source symbols transmitted are  $s_1$  followed by  $s_1$  or simply  $s_2$ .)

### The $n$ th Extension of a Block Code

- The  $n$ th extension of a block code, which maps the source symbol  $\underline{s_i}$  into the codeword  $\underline{A_i}$ , is a block code that maps the sequences of source symbols  $\underline{s_{i1}s_{i2}\cdots s_{in}}$

into the sequences of codewords  
 $\underline{A_{i1}A_{i2}\cdots A_{in}}$ .

- **A Necessary and Sufficient Condition of Block Codes' Unique Decodability**

A block code is uniquely decodable if and only if the  $n$ th extension of the code is nonsingular for every finite  $n$ .

Table 5. 5 The second extension of the nonsingular block code shown in Example 5.4

Source Symbol	Codeword	Source Symbol	Codeword
$S_1 S_1$	1 1	$S_3 S_1$	0 0 1
$S_1 S_2$	1 1 1	$S_3 S_2$	0 0 1 1
$S_1 S_3$	1 0 0	$S_3 S_3$	0 0 0 0
$S_1 S_4$	1 0 1	$S_3 S_4$	0 0 0 1
$S_2 S_1$	1 1 1	$S_4 S_1$	0 1 1
$S_2 S_2$	1 1 1 1	$S_4 S_2$	0 1 1 1
$S_2 S_3$	1 1 0 0	$S_4 S_3$	0 1 0 0
$S_2 S_4$	1 1 0 1	$S_4 S_4$	0 1 0 1

### 5.1.2.3 Instantaneous Codes

#### Definition of Instantaneous Codes

A uniquely decodable code is said to be instantaneous if it is possible to decode each codeword in a code symbol sequence without knowing the succeeding codewords.

Table 5.6 Three uniquely decodable codes

Source Symbol	Code 1	Code 2	Code 3
S <sub>1</sub>	00	1	1
S <sub>2</sub>	01	01	10
S <sub>3</sub>	10	001	100
S <sub>4</sub>	11	0001	1000

#### Example 5.6

- Table 5.6: three uniquely decodable codes.
- The first one is in fact a two-bit NBC.

In decoding we can immediately tell which source symbols are transmitted since each codeword has the same length.

- In the second code, code symbol “1” functions like a comma. Whenever we see a “1”, we know it is the end of the codeword.
- The third code is different from the previous two codes in that if we see a “10” string we are not sure if it corresponds to  $S_2$  until we see a succeeding “1”. Specifically, if the next code symbol is “0”, we still cannot tell if it is  $S_3$  since the next one may be “0” (hence  $S_4$ ) or “1” (hence  $S_3$ ). In this example, the next “1” belongs to the succeeding codeword. Therefore we see that code 3 is uniquely decodable. It is not instantaneous, however.

### Definition of the jth Prefix

Assume a codeword  $A_i = a_{i1}a_{i2} \cdots a_{ik}$ . Then the sequences of code symbols  $a_{i1}a_{i2} \cdots a_{ij}$

with  $1 \leq j \leq k$  is the  $j$ th order prefix of the codeword  $A_i$ .

### Example 5.7

- If a codeword is 11001, it has the following five prefixes: 11001, 1100, 110, 11, 1.
- The first order prefix is 1, while the fifth order prefix is 11001.

## A Necessary and Sufficient Condition of Being Instantaneous Codes

- ❖ A code is instantaneous if and only if no codeword is a prefix of some other codeword.
- ❖ This condition is called the *prefix condition*. Hence, the instantaneous code is also called the prefix condition code or sometimes simply the prefix code.
- ❖ In many applications, we need a block code that is nonsingular, uniquely decodable, and instantaneous.

### 5.1.2.4 Compact Code

- ◆ A uniquely decodable code is said to be compact if its average length is the minimum among all other uniquely decodable codes based on the same source alphabet  $S$  and code alphabet  $A$ .
- ◆ A compact code is also referred to as a *minimum-redundancy* code, or an *optimum* code.

### 5.1.3 Discrete Memoryless Sources

- The simplest model of an information source.
- In this model, the symbols generated by the source are **independent** of each other. That is, the source is memoryless or it has a zero-memory.

- Consider the information source expressed in Equation 5.1 as a discrete memoryless source.
  - The occurrence probabilities of the source symbols can be denoted by  $p(s_1), p(s_2), \dots, p(s_m)$ .
  - The lengths of the codewords can be denoted by  $l_1, l_2, \dots, l_m$ .
  - The average length of the code is then equal to

$$L_{avg} = \sum_{i=1}^m l_i p(s_i). \quad (5.1)$$

#### 5.1.4 Extensions of a Discrete Memoryless Source

- Instead of coding each source symbol in a discrete source alphabet, it is often useful to code blocks of symbols.

### 5.1.4.1 Definition

- Consider the zero-memory source alphabet  $S = \{s_1, s_2, \dots, s_m\}$ .
- If  $n$  symbols are grouped into a block, then there are a total of  $m^n$  blocks. Each block is considered as a new source symbol.
- These  $m^n$  blocks thus form an information source alphabet, called the  $n$ th extension of the source  $S$ , which is denoted by  $S^n$ .

### 5.1.4.2 Entropy

➤ Let each block be denoted by  $\mathbf{b}_i$  and

$$\mathbf{b}_i = (s_{i1}, s_{i2}, \dots, s_{in}). \quad (5.2)$$

➤ Because of the memoryless assumption, we have

$$p(\mathbf{b}_i) = \prod_{j=1}^n p(s_{ij}). \quad (5.3)$$

➤ Hence

$$H(S^n) = n \cdot H(S). \quad (5.4)$$

### 5.1.4.3 Noiseless Source Coding Theorem

- ❖ For a discrete zero-memory information source  $S$ , Shannon's noiseless coding theorem can be expressed as

$$H(S) \leq L_{avg} < H(S) + 1; \quad (5.5)$$

that is, there exists a variable-length code whose average length is bounded below by the entropy of the source (that is encoded) and bounded above by the entropy plus 1.

- ❖ Since the  $n$ th extension of the source alphabet,  $S^n$ , is itself a discrete memoryless source, we can apply the above result to it.

$$H(S^n) \leq L_{avg}^n < H(S^n) + 1, \quad (5.6)$$

where  $L_{avg}^n$  is the average codeword length of a variable-length code for the  $S^n$ .

- ❖ Since  $H(S^n) = nH(S)$  and  $L_{avg}^n = nL_{avg}$ , we have

$$H(S) \leq L_{avg} < H(S) + \frac{1}{n}. \quad (5.7)$$

- ❖ Therefore, when coding blocks of  $n$  source symbols, the noiseless source coding theory states that for an arbitrary positive number  $\mathbf{e}$ , there is a variable-length code which satisfies the following:

$$H(S) \leq L_{avg} < H(S) + \mathbf{e} \quad (5.8)$$

- ❖ as  $n$  is large enough. To make  $\mathbf{e}$  arbitrarily small, we have to make the block size  $n$  large enough.
- ❖ In most of cases, the price turns out to be small when  $n$  is not larger enough.

## 5.2 Huffman Codes

- In many cases, we need a direct encoding method that is optimum and instantaneous (hence uniquely decodable) for an information source with finite source symbols in source alphabet  $S$ .
- Huffman code is the first such optimum code [huffman 1952].
- The most frequently used at present.
- It can be used for  $r$ -ary encoding as  $r > 2$ . For the notational brevity, however, we discuss only the Huffman coding used in the binary case presented here.

### 5.2.1 Required Rules for Optimum Instantaneous Codes

- ◆ Consider an information source:

$$S = (s_1, s_2, \dots, s_m). \quad (5.9)$$

- ◆ WLOG, assume the occurrence probabilities of the source symbols are as follows:

$$p(s_1) \geq p(s_2) \geq \cdots \geq p(s_{m-1}) \geq p(s_m)$$

- ◆ Since we are seeking the optimum code for  $S$ , the lengths of codewords assigned to the source symbols should be

$$l_1 \leq l_2 \leq \cdots \leq l_{m-1} \leq l_m. \quad (5.10)$$

- ◆ Based on the requirements of the optimum and instantaneous code, Huffman derived the following rules (restrictions):

1.  $l_1 \leq l_2 \leq \cdots \leq l_{m-1} = l_m.$  (5.11)

2. The codewords of the two least probable source symbols should be the same except for their last bits.

3. Each possible sequence of length  $l_m - 1$  bits must be used either as a codeword or must have one of its prefixes used as a codeword.

## 5.2.2 Huffman Coding Algorithm

- Based on these three rules, we see that the two least probable source symbols have equal-length codewords.
- These two codewords are identical except for the last bits, the binary 0 and 1, respectively.
- Therefore, these two source symbols can be combined to form a single new symbol.
- Its occurrence probability is the sum of two source symbols, i.e.,  $p(s_{m-1}) + p(s_m)$ .
- Its codeword is the common prefix of order  $l_m - 1$  of the two codewords assigned to  $s_m$  and  $s_{m-1}$ , respectively.
- The new set of source symbols thus generated is referred to as the first auxiliary source alphabet, which is one source symbol less than the original source alphabet.
- In the first auxiliary source alphabet, we can rearrange the source symbols according to a nonincreasing order of their occurrence probabilities.

- The same procedure can be applied to this newly created source alphabet.
- The second auxiliary source alphabet will again have one source symbol less than the first auxiliary source alphabet.
- The procedure continues. In some step, the resultant source alphabet will have only two source symbols. At this time, we combine them to form a single source symbol with a probability of 1. The coding is then complete.

## **Example 5.9**

Table 5. 7 The source alphabet, and Huffman codes in Example 5.9

Source symbol	Occurrence probability	Codeword assigned	Length of Codeword
S <sub>1</sub>	0.3	00	2
S <sub>2</sub>	0.1	101	3
S <sub>3</sub>	0.2	11	2
S <sub>4</sub>	0.05	1001	4
S <sub>5</sub>	0.1	1000	4
S <sub>6</sub>	0.25	01	2

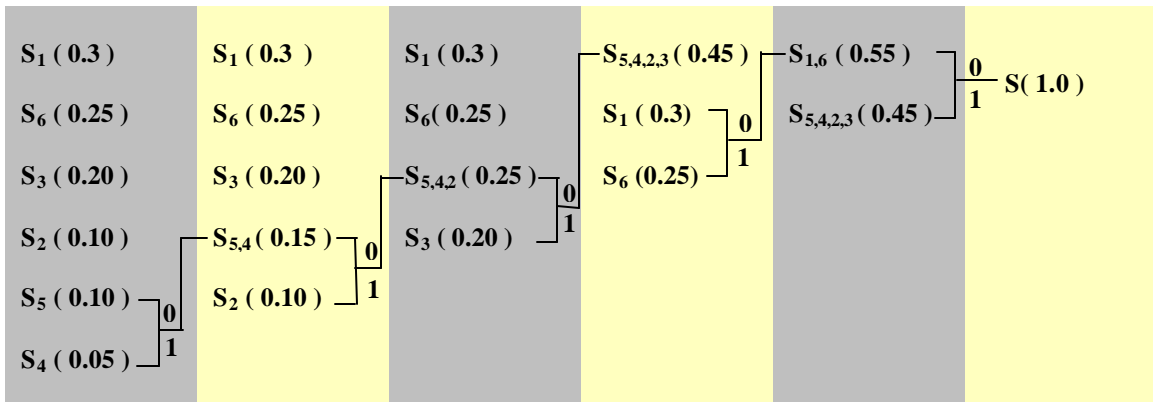


Figure 5.1 Huffman coding procedure in Example 5.9

### 5.2.2.1 Applications

- Recall that it has been used in differential coding and transform coding.
- In TC, the magnitude of the quantized nonzero transform coefficients and the run-length of zeros in the zigzag scan are encoded by using the Huffman code.

### 5.2.3 Modified Huffman Codes

when the occurrence probabilities are skewed, and the number of less probable source symbols is large, the required codebook memory will be large.

The modified Huffman code can reduce the memory requirement while keeping almost the same optimality.

## 5.3 Arithmetic Codes

Arithmetic coding:

- ✓ quite different from Huffman coding
- ✓ gaining increasing popularity

### 5.3.1 Limitations of Huffman Coding

- Huffman coding is optimum for block-encoding a source alphabet, with each source symbol having an occurrence probability.
- The average codeword length achieved by Huffman coding satisfies the following inequality [gallagher 1978].

$$H(S) \leq L_{avg} < H(S) + p_{\max} + 0.086$$

(5.22)

$p_{\max}$ : the maximum occurrence probability in the set of the source symbols.

- In the case where the probability distribution among source symbols is **skewed** (some probabilities are small, while some are quite large), the upper bound may be large, implying that the coding redundancy may not be small.
- An extreme situation. There are only two source symbols. One has a very small probability, while the other has a very large probability (very close to 1).
  - Entropy of source alphabet is close to 0 since the uncertainty is very small.
  - Using Huffman coding, however, we need two bits: one for each.  $\Rightarrow$ 
    - ✓ average codeword length is 1
    - ✓ redundancy  $h \approx 1$
    - ✓ This agrees with Equation 5.22.
    - ✓ This inefficiency is due to the fact that Huffman coding always encodes a source symbol with an integer number of bits.

- The fundamental idea behind Huffman coding is **block coding**.
  - That is, some codeword having an integral number of bits is assigned to a source symbol.
  - A message may be encoded by cascading the relevant codewords. It is the *block-based* approach that is responsible for the limitations of Huffman codes.
- Another limitation is that when encoding a message that consists of a sequence of source symbols **the *n*th extension Huffman coding** needs to enumerate all possible sequences of source symbols having the same length, as discussed in coding the *n*th extended source alphabet. This is not computationally efficient.
- ◆ Quite different from Huffman coding, arithmetic coding is *stream-based*. It overcomes the drawbacks of Huffman coding.

- ◆ A string of source symbols is encoded as a string of code symbols. Hence free of the integral-bits-per-source-symbol restriction and more efficient.
- ◆ Arithmetic coding may reach the theoretical bound to coding efficiency specified in the noiseless source coding theorem for any information source.

### 5.3.2 The Principle of Arithmetic Coding

#### Example 5.12

- Same source alphabet as that used in Example 5.9.
- In this example, however, a string of source symbols  $s_1s_2s_3s_4s_5s_6$  is encoded.

Table 5.8 Source alphabet and cumulative probabilities in Example 5.12

Source symbol	Occurrence probability	Associated subintervals	CP
$S_1$	0.3	[ 0, 0.3 )	0
$S_2$	0.1	[ 0.3, 0.4 )	0.3
$S_3$	0.2	[ 0.4, 0.6 )	0.4
$S_4$	0.05	[ 0.6, 0.65 )	0.6
$S_5$	0.1	[ 0.65, 0.75 )	0.65
$S_6$	0.25	[ 0.75, 1.0 )	0.75

### 5.3.2.1 Dividing Interval [0,1) into Subintervals

#### ❖ *Cumulative probability* (CP)

Slightly different from that of cumulative distribution function (CDF) in probability theory.

$$CDF(s_i) = \sum_{j=1}^i p(s_j). \quad (5.12)$$

$$CP(s_i) = \sum_{j=1}^{i-1} p(s_j), \quad (5.13)$$

where  $CP(s_1) = 0$ .

❖ Each subinterval

- Its lower end point located at  $CP(s_i)$ .
- Width of each subinterval equal to probability of corresponding source symbol.
- A subinterval can be completely defined by its lower end point and its width.
- Alternatively, it is determined by its two end points: the lower and upper end points (sometimes also called the left and right end points).

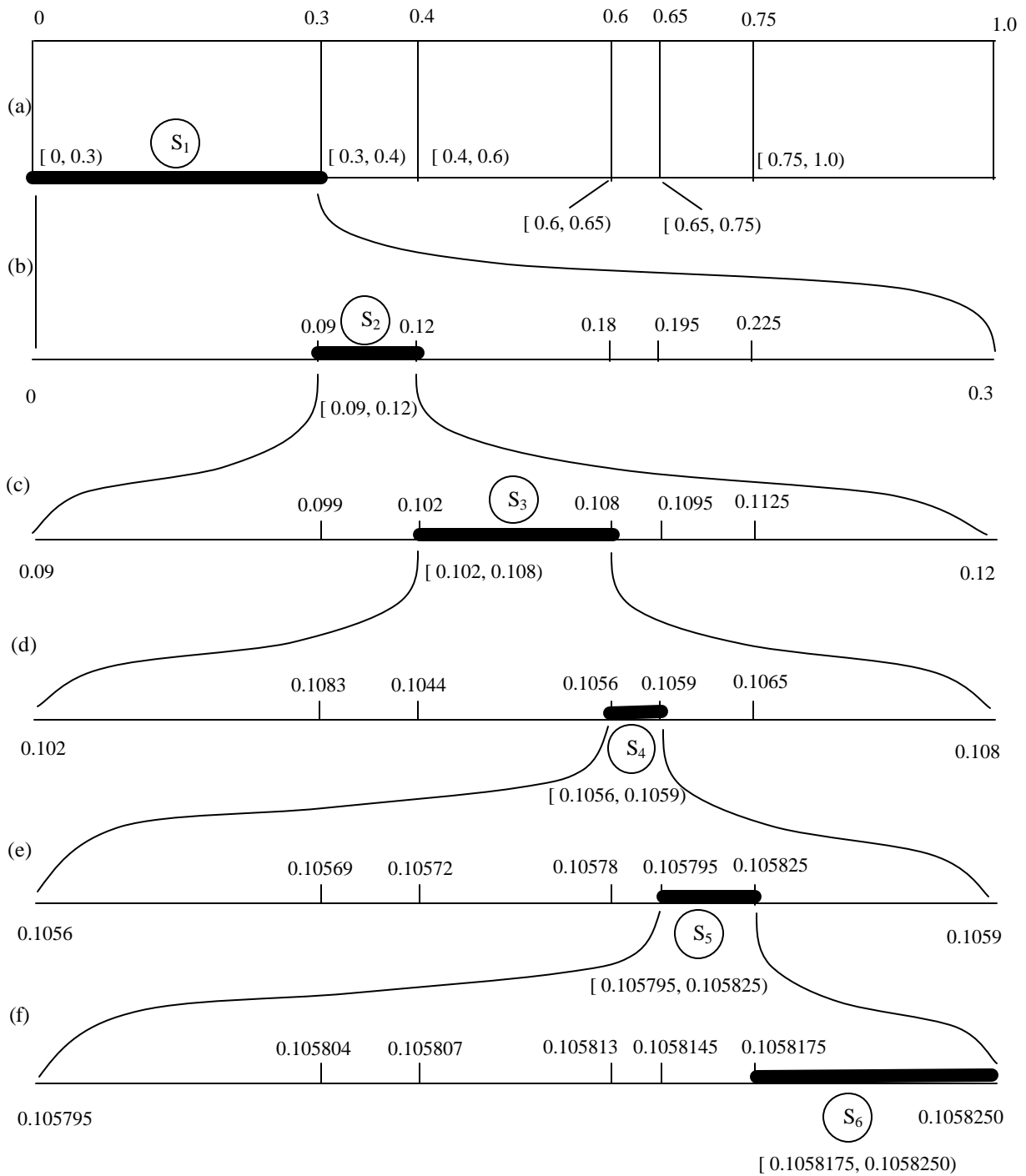


Figure 5. 2 Arithmetic coding working on the same source alphabet as that in Example 5.9. The encoded symbol string is  $S_1 S_2 S_3 S_4 S_5 S_6$ .

### 5.3.2.2 Encoding

#### Encoding the First Source Symbol

Refer to Part (a) of Figure 5.3. Since the first symbol is  $s_1$ , we pick up its subinterval  $[0, 0.3)$ . Picking up the subinterval  $[0, 0.3)$  means that any real number in the subinterval, i.e., any real number equal to or greater than 0 and smaller than 0.3, can be a pointer to the subinterval, thus representing the source symbol  $s_1$ . This can be justified by considering that all the six subintervals are disjoint.

#### Encoding the Second Source Symbol

- Refer to Part (b) of Figure 5.3. We use the same procedure as used in Part (a) to divide the interval  $[0, 0.3)$  into six subintervals. Since the second symbol to be encoded is  $s_2$ , we pick up its subinterval  $[0.09, 0.12)$ .
- Notice that the subintervals are recursively generated from Part (a) to Part (b). It is known that an interval may be completely specified by its lower end point and width. Hence, the subinterval recursion in the

arithmetic coding procedure is equivalent to the following two recursions: **end point recursion** and **width recursion**.

✓ The lower end point recursion:

$$L_{new} = L_{current} + W_{current} \cdot CP_{new}$$

$L$  : the lower end points

$W$  : the width

*new*: the new recursion

*current*: the current recursion

✓ The width recursion is

$$W_{new} = W_{current} \cdot p(s_i), \quad (5.14)$$

Encoding the Third Source Symbol

Encoding the Fourth, Fifth and Sixth Source Symbols

- The resulting subinterval  $[0.1058175, 0.1058250)$  can represent the source symbol string  $s_1 s_2 s_3 s_4 s_5 s_6$ .

### 5.3.2.3 Decoding

- Theoretically, any real numbers in the interval can be the code string for the input symbol string since all subintervals are disjointed.
- Often, however, the lower end of the final subinterval is used as the code string.
- Now let us examine how the decoding process is carried out with the lower end of the final subinterval.
- ◆ The decoder knows the encoding procedure and therefore has the information contained in Part (a) of Figure 5.3.

Since  $0 < 0.1058175 < 0.3$ .

the symbol  $S_1$  is first decoded.

- Once the first symbol is decoded, the decoder may know the partition of subintervals shown in Part (b) of Figure 5.3. It is then determined that

$$0.09 < 0.1058175 < 0.12.$$

That is, the lower end is contained in the subinterval corresponding to the symbol  $S_2$ .

As a result,  $S_2$  is the second decoded symbol.

The procedure repeats itself until all six symbols are decoded.

- Note that a terminal symbol is necessary to inform the decoder to stop decoding.
- The above procedure gives us an idea of how decoding works.
- The decoding process, however, does not need to construct Parts (b), (c), (d), (e) and (f) of Figure 5.3.
- Instead, the decoder only needs the information contained in Part (a) of Figure 5.3.
- Decoding can be split into the following three steps: *comparison*, *readjustment* (subtraction), and *scaling* [Langdon 1984].

- In summary, considering the way in which Parts (b), (c), (d), (e) and (f) of Figure 5.3 are constructed, we see that the three steps discussed in the decoding process: comparison, readjustment and scaling exactly “undo” what the encoding procedure has done.
- Back to Example:
  - After symbol  $s_1$  is first decoded,
  - the lower end of the corresponding interval, 0, is subtracted from 0.1058175, resulting in 0.1058175.
  - This value is then divided by  $p_1=0.3$ , resulting in 0.352725. Compare this value with Part (a), we decode  $s_2$  since  $0.3 < 0.352725 < 0.4$ .
  - We can decode the rest in the same way.

### 5.3.2.4 Observations

- ◆ Both encoding and decoding involve only arithmetic operations (addition and multiplication in encoding, subtraction and division in decoding). This explains the name *arithmetic coding*.
- ◆ We see that an input source symbol string  $s_1s_2s_3s_4s_5s_6$ , via encoding, corresponds to a subinterval  $[0.1058175, 0.1058250)$ . Any number in this interval can be used to denote the string of the source symbols.
- ◆ We also observe that arithmetic coding can be carried out in an *incremental* manner. That is, source symbols are fed into the encoder one by one and the final subinterval is refined continually, i.e., the code string is generated continually.
- ◆ Furthermore, it is done in a manner called *first in first out* (FIFO). That is, the source symbol encoded first is decoded first.
- ◆ It is obvious that the width of the final subinterval becomes smaller and smaller

when the length of the source symbol string becomes larger and larger. This causes what is known as the **precision problem**.

It is this problem that prohibited arithmetic coding from practical usage for quite a long period of time.

Only after this problem was solved in the late 1970s, did arithmetic coding become an increasingly important coding technique.

- ◆ It is necessary to have a termination symbol at the end of an input source symbol string. In this way, an arithmetic coding system is able to know when to terminate decoding.
- ◆ To encode the same source symbol string, Huffman coding can be implemented in two different ways.

One way is shown in Example 5.9. We construct a fixed codeword for each source symbol. Since Huffman coding is instantaneous, we can cascade the corresponding codewords to form the output, a 17-bit code string 00.101.11.1001.1000.01, where the five

periods are used to indicate different codewords for easy reading.

- ◆ As we see that for the same source symbol string, the final subinterval obtained by using arithmetic coding is  $[0.1058175, 0.1058250)$ . It is noted that the 15-bit binary decimal,  $0.000110111111111$ , is equal to the decimal  $0.1058211962$ , which falls into the final subinterval representing the string  $s_1s_2s_3s_4s_5s_6$ .
- ◆ This indicates that the arithmetic coding is more efficient than the Huffman coding in this example.
- ◆ Another way is to form a  $6^{\text{th}}$  extension of the source alphabet as discussed in Section 5.1.4: treat each group of six source symbols as a new source symbol; calculate its occurrence probability by multiplying the related six probabilities; then apply the Huffman coding algorithm to the  $6^{\text{th}}$  extension of the discrete memoryless source. This is called the  $6^{\text{th}}$  extension of Huffman block code. In other words, in order to

encode the source string  $s_1s_2s_3s_4s_5s_6$ , (the 6<sup>th</sup> extension of) Huffman coding encodes all of the  $6^6=46656$  codewords in the 6<sup>th</sup> extension of the source alphabet. This implies a high complexity in implementation and a large codebook, hence not efficient.

- ◆ Similar to the case of Huffman coding, arithmetic coding is also applicable to r-ary encoding with  $r > 2$ .

### 5.3.3 Implementation Issues

- The **growing precision problem**.
- This problem has been resolved and the **finite precision** arithmetic is now used in arithmetic coding.
- This advance is due to the **incremental implementation** of arithmetic coding.

#### 5.3.3.1 Incremental Implementation

- We observe that after the third symbol,  $s_3$ , is encoded, the resultant subinterval is  $[0.102, 0.108)$ .
  - ✓ That is, the two most significant decimal digits are the same and they remain the same in the encoding process.
  - ✓ We can transmit these two digits without affecting the final code string.
- After the fourth symbol  $s_4$  is encoded, the resultant subinterval is  $[0.1056, 0.1059)$ .
  - ✓ One more digit, 5, can be transmitted.
  - ✓ After the sixth symbol is encoded, the final subinterval is  $[0.1058175, 0.1058250)$ .
  - ✓ The cumulative output is 0.1058. Refer to Table 5.11.
- This important observation reveals that we are able to incrementally transmit output (the code symbols) and receive input (the source symbols that need to be encoded).

- Table 5. 9 Final subintervals and cumulative output in Example 5.12

Source symbol	Final subinterval		Cumulative output
	Lower end	Upper end	
$S_1$	0	0.3	—
$S_2$	0.09	0.12	—
$S_3$	0.102	0.108	0.10
$S_4$	0.1056	0.1059	0.105
$S_5$	0.105795	0.105825	0.105
$S_6$	0.1058175	0.1058250	0.1058

### 5.3.3.2 Other Issues

#### Eliminating Multiplication

#### Carry-Over Problem

### 5.3.4 History

- The idea of encoding by using cumulative probability in some ordering, and decoding by comparison of magnitude of binary

fraction was introduced in **Shannon's** celebrated paper [shannon 1948].

- The recursive implementation of arithmetic coding was devised by **Elias** (another member in Fano's first information theory class at MIT).

This unpublished result was first introduced by Abramson as a note in his book on information theory and coding [abramson 1963].

- The result was further developed by Jelinek in his book on information theory [jelinek 1968].
- The growing precision problem prevented arithmetic coding from practical usage, however. The proposal of using finite precision arithmetic was made independently by **Pasco** [pasco 1976] and **Rissanen** [rissanen 1976].
- Practical arithmetic coding was developed by **several independent groups** [rissanen 1979, rubin 1979, guazzo 1980].

- A well-known tutorial paper on arithmetic coding appeared in [**langdon** 1984].
- The tremendous efforts made in **IBM** lead to a new form of adaptive binary arithmetic coding known as the Q-coder [pennebaker 1988].
- Based on the Q-coder, the activities of **JPEG and JBIG** combined the best features of the various existing arithmetic coders and developed the binary arithmetic coding procedure known as the QM-coder [pennebaker 1992].

### 5.3.5 Applications

- Arithmetic coding is becoming popular.
- Note that **in text and bilevel image applications** there are only two source symbols (black and white), and the occurrence probability is skewed. Therefore binary arithmetic coding achieves high coding efficiency.

- It has been successfully applied to bilevel image coding [langdon 1981] and adopted by the international standards for bilevel image compression JBIG.
- It has also been adopted by the JPEG.

## 5.4 References

[abramson 1963] N. Abramson, *Information Theory and Coding*, New York: McGraw-Hill, 1963.

[bell 1990] T. C. Bell, J. G. Cleary and I. H. Witten, *Text Compression*, Englewood, NJ: Prentice Hall, 1990.

[blahut 1986] R. E. Blahut, *Principles and Practice of Information Theory*, Reading MA: Addison-Wesley, 1986.

[fano 1949] R. M. Fano, "The transmission of information," *Technical Report 65*, Research Laboratory of Electronics, MIT, Cambridge, MA, 1949.

[gallagher 1978] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Transactions on Information Theory*, vol. IT-24, no. 6, pp. 668-674, November 1978.

[guazzo 1980] M. Guazzo, "A general minimum-redundancy source-coding algorithm," *IEEE Transactions on Information Theory*, vol. IT-26, no. 1, pp. 15-25, January 1980.

[hankamer 1979] M. Hankamer, "A modified Huffman procedure with reduced memory requirement," *IEEE Transactions on Communications*, vol. COM-27, no. 6, pp. 930-932, June 1979.

[huffman 1952] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of The IRE*, vol. 40, pp. 1098-1101, September 1952.

- [jelinek 1968] F. Jelinek, *Probabilistic Information Theory*, New York: McGraw-Hill, 1968.
- [langdon 1981] G. G. Langdon, Jr. and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Transactions on Communications*, vol. COM-29, no. 6, pp. 858-867, June 1981.
- [langdon 1982] G. G. Langdon, Jr. and J. Rissanen, "A simple general binary source code," *IEEE Transactions on Information Theory*, IT-28, 800 (1982).
- [langdon 1984] G. G. Langdon, Jr., "An introduction to arithmetic coding," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135-149, March 1984.
- [nelson 1996] M. Nelson and J. Gailly, *The Data Compression Book*, second edition, New York: M&T Books, 1996.
- [pasco 1976] R. Pasco, *Source Coding Algorithms for Fast Data Compression*, Ph.D. dissertation, Stanford University, 1976.
- [pennebaker 1988] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr. and R. B. Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic Coder," *IBM Journal of Research and Development*, vol. 32, no. 6, pp. 717-726, November 1988.
- [pennebaker 1992] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard*, New York: Van Nostrand Reinhold, 1992.
- [rissanen 1976] J. J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM Journal of Research and Development*, vol. 20, pp. 198-203, May 1976.
- [rissanen 1979] J.J. Rissanen and G. G. Landon, "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149-162, March 1979.
- [rubin 1979] F. Rubin, "Arithmetic stream coding using fixed precision registers," *IEEE Transactions on Information Theory*, vol. IT-25, no. 6, pp. 672-675, November 1979.
- [sayood 1996] K. Sayood, *Introduction to Data Compression*, San Francisco, CA: Morgan Kaufmann Publishers, 1996.
- [shannon 1948] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379-423 (Part I), July 1948, pp. 623-656 (Part II), October 1948. decodability