Chapter 6

RUN-LENGTH AND DICTIONARY CODING

----- *INFORMATION THEORY RESULTS (III)*

6.1 Markov Source Model

- In reality, many sources are dependent in nature.
  - Namely, source has memory: previous status has an influence on present status.
  - Example: interpixel correlation in digital images.
  - Therefore it is necessary to introduce models that can reflect this type of dependence.
  - A Markov source model is often used.

6.1.1 Discrete Markov Source

- Consider: a source alphabet

$$S = \{s_1, s_2, \cdots, s_m\}$$

with occurrence probability p

- An *l*th order Markov source:

$$p(s_j \mid s_{i1}, s_{i2}, \cdots, s_{il}, \cdots) = p(s_j \mid s_{i1}, s_{i2}, \cdots, s_{il})$$

$$j, i1, i2, \cdots, il, \cdots \in \{1, 2, \cdots, m\}.$$

This equation states that:

✓ source symbols not independent of each other

✓ occurrence probability of a source symbol is determined by some of its previous symbols.

- ✓knowledge of entire sequence of previous symbols is equivalent to that of <u>the l symbols immediately preceding the current symbol</u>.

- An l-th order Markov source can be described by a ***state diagram***.
  - ✓ A state is a sequence of $(s_{i1}, s_{i2}, \cdots, s_{il})$
  - ✓any group of l symbols from the m symbols in the S forms a state.
  - ✓an lth order Markov source with m symbols in the S has a total of m^l different states.

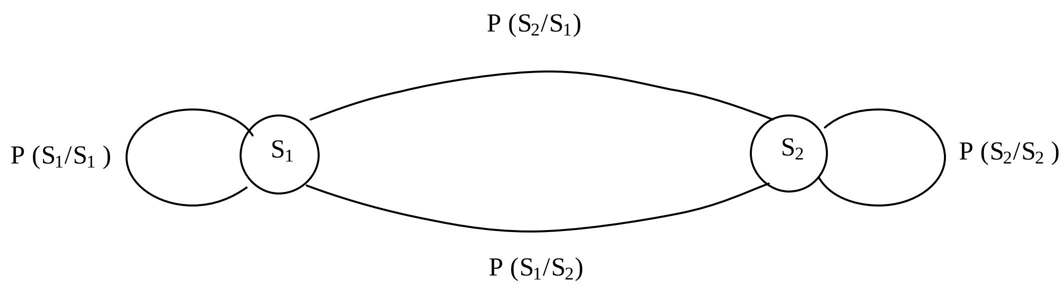
- <u>The source entropy at a state</u> $(s_{i1}, s_{i2,} \cdots, s_{il})$ is defined as

$$H(S \mid s_{i1}, s_{i2}, \cdots, s_{il}) = -\sum_{j=1}^{m} p(s_j \mid s_{i1}, s_{i2}, \cdots, s_{il}) \log_2 p(s_j \mid s_{i1}, s_{i2}, \cdots, s_{il})$$

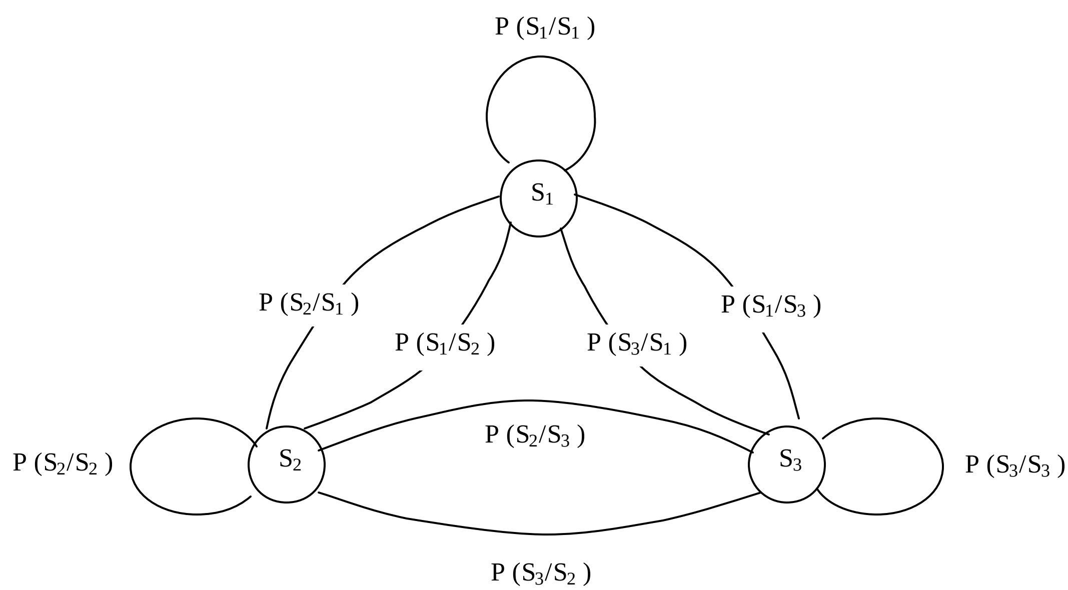
- The source entropy is defined as the statistical average of the entropy at all the states. That is,

$$H(S) = \sum_{(s_{i1}, s_{i2}, \cdots, s_{il}) \in S^l} p(s_{i1}, s_{i2}, \cdots, s_{il}) H(S \mid s_{i1}, s_{i2}, \cdots, s_{il})$$

S^l: the lth extension of the S.



(a)



(b)

Figure 6.1 State diagrams of the first order Markov sources with their source alphabets having (a) Two symbols (b) Three symbols

6.1.1 Extensions of a Discrete Markov Source

6.1.1.1 Definition

- Consider an *l*-th order Markov source $S = \{s_1, s_2, \cdots, s_m\}$ and a set of conditional probabilities $p(s_j \mid s_{i1}, s_{i2}, \cdots, s_{il})$, where $j, i1, i2, \cdots, il \in \{1, 2, \cdots, m\}$.

  Similar to discrete memoryless source, if *n* symbols are grouped into a block, then there are a total of m^n blocks.

- Each block can be viewed as a new source symbol. Hence, these m^n blocks form a new information source alphabet, called the *n*th extension of the source *S* and denoted by S^n.

- The *n*th extension of the *l*-th order Markov source is a *n*-th order Markov source,

$$k = \left\lceil \frac{l}{n} \right\rceil, \tag{6. 1}$$

where the notation $\lceil a \rceil$ represents the operation of taking the smallest integer greater than or equal to the quantity a.

6.1.1.2 Entropy [abramson 1963]

$$H(S^n) = nH(S) \tag{6. 2}$$

6.1.2 Autoregressive (AR) Model

- ♦ AR model: another kind of dependent source model that has been used often in image coding.
- ♦ It is defined below

$$s_j = \sum_{k=1}^{l} a_k s_{ik} + x_j, \tag{6. 3}$$

s_j: currently observed source symbol,

s_{ik} with $k = 1, 2, \cdots, l$: the l preceding observed symbols,

a_k's: coefficients,

x_j: the current input to the model.

6.2 Run-Length Coding (RLC)

- ***run*** : the repetition of a symbol.
- ***run-length*:** number of repeated symbols.
- Instead of encoding the consecutive symbols, it is more efficient to encode the run-length and the value that these consecutive symbols commonly share.
- According to an excellent early review on binary image compression [arps 1979], RLC has been in use since the earliest days of information theory [shannon 1949, laemmel 1951].

♦ Application:

- ➢Adopted in JPEG (multi-level image coding)
- ➢Binary document coding
- ➢Adopted in facsimile coding standards: the CCITT Recommendations T.4 and T.6.

♦ Classification:

- RLC using only the horizontal correlation between pixels on the same scan line is called <u>1-D RLC</u>.

  Noted that the first order <u>Markov source model with two symbols</u> in the source alphabet depicted in Figure 6.1 (a) can be used to characterize 1-D RLC.

- To achieve higher coding efficiency, <u>2-D RLC</u> utilizes both horizontal and vertical correlation between pixels.

6.2.1 1-D Run-Length Coding

- In this technique, each scan line is encoded independently.
- Each scan line can be considered as a sequence of alternating, independent white runs and black runs.
- As an agreement between encoder and decoder, the first run in each scan line is assumed to be a white run.

  If the first actual pixel is black, then the run-length of the first white run is set to be zero.
- At the end of each scan line, there is a special codeword called end-of-line (EOL). The decoder knows the end of a scan line when it encounters an EOL codeword.

♦ Denote run-length by r, which is integer-valued. All of the possible run-lengths construct a source alphabet R, which is a random variable. That is,

$$R = \{r : r \in 0, 1, 2, \cdots\}. \tag{6. 4}$$

- Measurements on typical binary documents have shown that the maximum compression ratio is about 25% higher when the white and black runs are encoded separately [hunter 1980].

- Huffman coding is then applied to two source alphabets.

  - According to CCITT T.4, A4 size (210×297 mm) documents should be accepted by facsimile machines.

  - In each scan line, there are 1728 pixels. This means $m = 1728$.

  - Two source alphabets of such a large size imply the requirement of two large codebooks, hence the requirement of large storage space.

- Therefore, some modification was made, resulting in **the "modified" Huffman (MH) code.**

- In the modified Huffman code, if the run-length is larger than 63, then the run-length is represented as

$$r = M \times 64 + T \quad as \quad r > 63$$

✓ M : integer values from 1,2, to 27,

✓ $M \times 64$: the makeup run-length;

✓ T : integer values from 0, 1 to 63, and called the terminating run-length.

- That is, if $r > 63$, the run-length is represented by a makeup codeword and a terminating codeword.
- If $r \leq 63$, the run-length is represented by a terminating codeword only.
- In this way, the requirement of large storage space is alleviated.

Table 6. 1 Modified Huffman code table [hunter 1980]

Run-length	White runs	Black runs
	Terminating Codewo rds	
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
4	1011	011
5	1100	0011
6	1110	0010
7	1111	00011
8	10011	000101
⋮	⋮	⋮
60	01001011	000000101100
61	00110010	000001011010
62	00110011	000001100110
63	00110100	000001100111
	Make-up Codewords	
64	11011	0000001111
128	10010	000011001000
192	010111	000011001001
256	0110111	000001011011
⋮	⋮	⋮
1536	010011001	0000001011010
1600	010011010	0000001011011
1664	011000	0000001100100
1728	010011011	0000001100101
EOL	000000000001	000000000001

6.2.1 2-D Run-Length Coding

- In CCITT T.4, the modified relative element address designate (READ) code, also known as the modified READ code or simply the **MR** code, is adopted.
- The modified READ code operates in a line-by-line manner.
- In Figure 6.2, two lines are shown.
    - The top line is called the reference line, which has been coded.
    - The bottom line is referred to as the coding line, which is being coded.
    - There are a group of five changing pixels, a_0, a_1, a_2, b_1, b_2, in the two lines.
    - Their relative positions decide which of the three coding modes is used.

- The starting changing pixel a_0 (hence, five changing points) moves from left to right and from top to bottom as 2-D run-length coding proceeds.

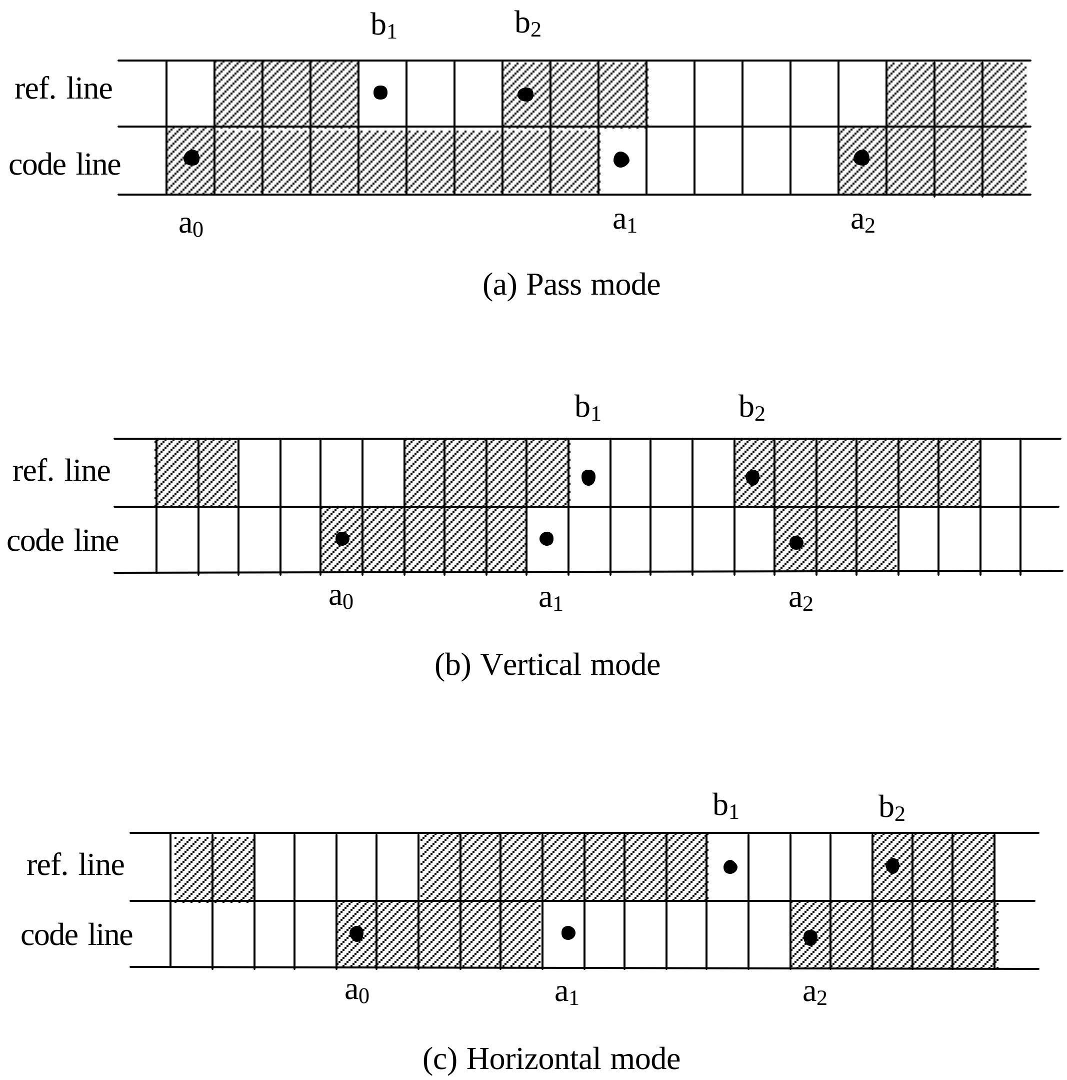


Figure 6.2 2-D run-length coding

6.2.1.1 Five Changing Pixels

- A changing pixel: the first pixel encountered in white or black runs when we scan an image line-by-line, from left to right, and from top to bottom.
- The five changing pixels are:
  - ❖ a_0: The reference changing pixel in the coding line. Its position is defined in the previous coding mode, whose meaning will be explained shortly. At the beginning of a coding line, a_0 is an imaginary white changing pixel located before the first actual pixel in the coding line.
  - ❖ a_1: The next changing pixel in the coding line. Because of the above-mentioned left-to-right and top-to-bottom scanning order, it is at the right-hand side of a_0. Since it is a

changing pixel, it has an opposite "color" to that of a_0.

- ❖ a_2: The next changing pixel after a_1 in the coding line. It is to the right of a_1 and has the same color as that of a_0.
- ❖ b_1: The changing pixel in the reference line that is closest to a_0 from the right and has the same color as a_1.
- ❖ b_2: The next changing pixel in the reference line after b_1.

6.2.1.2 Three Coding Modes

- ♦ **Pass Coding Mode**: If the changing pixel b_2 is located to the left of the changing pixel a_1, it means that the run in the reference line starting from b_1 is not adjacent to the run in the coding line starting from a_1. Note that these two runs have the same color. This is called pass coding mode.

A special codeword, “0001”, is sent out from transmitter. The receiver then knows that the run starting from a_0 in the coding line does not end at the pixel below b_2.

This pixel (below b_2 in the coding line) is identified as the reference changing pixel a_0 of the new set of five changing pixels for the next coding mode.

- **Vertical Coding Mode**: If the relative distance along the horizontal direction between the changing pixels a_1 and b_1 is not larger than 3 pixels, the coding is conducted in vertical coding mode.

  That is, the position of a_1 is coded with reference to the position of b_1. Seven different codewords are assigned to seven different cases: the distance between a_1 and b_1 equals 0,

± 1, ± 2, ± 3, where + means a_1 is to the right of b_1, while – means a_1 is to the left of b_1.

The a_1 then becomes the reference changing pixel a_0 of the new set of five changing pixels for the next coding mode.

- **Horizontal Coding Mode**: If the relative distance between the changing pixels a_1 and b_1 is larger than 3 pixels, the coding is conducted in horizontal coding mode.

  Here, 1-D RLC is applied. Specifically, the transmitter sends out a codeword consisting the following three parts: a flag "001"; a 1-D run-length codeword for the run from a_0 to a_1; a 1-D run-length codeword for the run from a_1 to a_2.

  The a_2 then becomes the reference changing pixel a_0 of the new set of five changing pixels for the next coding mode.

Table 6.2 2-D RLC table (from [hunter 1980]), $| x_i y_j |$: distance between x_i and y_j, $x_i y_j > 0$: x_i is right to y_j, $x_i y_j < 0$: x_i is left to y_j. $(x_i y_j)$: codeword of the run denoted by $x_i y_j$ taken from the modified Huffman code

Mode	Conditions	Output codeword	Position of New a_0
Pass coding mode	$b_2 a_1 < 0$	0001	under b_2 in coding line
Vertical coding mode	$a_1 b_1 = 0$	1	a_1
	$a_1 b_1 = 1$	011	
	$a_1 b_1 = 2$	000011	
	$a_1 b_1 = 3$	0000011	
	$a_1 b_1 = -1$	010	
	$a_1 b_1 = -2$	000010	
	$a_1 b_1 = -3$	0000010	
Horizontal coding mode	$\|a_1 b_1\| > 3$	$001+(a_0 a_1)+(a_1 a_2)$	a_2

6.2.2 Effect of Transmission Error and Uncompressed Mode

6.2.2.1 Error Effect in the 1-D RLC Case

- With the EOL, 1-D RLC encodes each scan line independently.
- If a transmission error occurs in a scan line, there are two possibilities that the effect caused by the error is limited within the scan line.
  - One possibility: ***resynchronization*** is established after a few runs.

    One example:

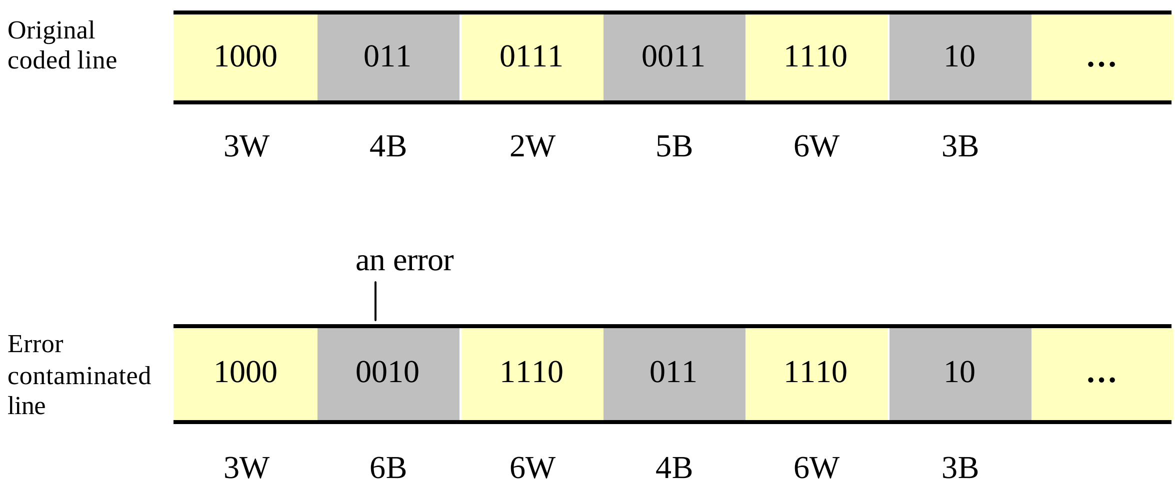


Figure 6. 3 Establishment of resynchronization after a few runs

    - Another possibility: the EOL forces resynchronization.

- In summary, the 1-D RLC will not propagate transmission error between scan lines.

- Though error detection and retransmission of data via an automatic repeat request (ARQ) system is supposed to be able to effectively handle the error susceptibility issue, the ARQ technique was not included into CCITT T.4 due to the computational complexity and extra transmission time required.

- Once the number of decoded pixels between two consecutive EOL codewords is not equal to 1728 (for an A4 ize document), an error has been identified. Some ***error concealment*** techniques can be used to reconstruct the scan line [hunter 1980].

    - For instance, we can repeat the previous line,

- or replace the damaged line by a white line,
- or use a correlation technique to recover the line as much as possible.

6.2.2.2 Error Effect in the 2-D RLC Case

- 2-D RLC:
  - More efficient than 1-D RLC
  - More susceptible to transmission errors than the 1-D RLC
  - To prevent error propagation, there is a parameter used in 2-D RLC, known as the ***K-factor***, which specifies <u>the number of scan lines that are 2-D RLC coded</u>.

    Recommendation T.4 defined that <u>no more than K-1</u> consecutive scan lines be 2-D RLC coded after a 1-D RLC coded line.

    - For binary documents scanned at normal resolution, K=2.

- For documents scanned at high resolution, K=4.

- According to [arps 1979], there are two different types of algorithms in binary image coding, ***raster*** algorithms and ***area*** algorithms.
  - Raster algorithms only operate on data within one or two raster scan lines. They are hence mainly 1-D in nature.
  - Area algorithms are truly 2-D in nature.
  - Area algorithms
    - require large memory space
    - susceptible to transmission noise.
  - From our discussion above, we see that both 1-D and 2-D RLC defined in T.4 belong to the category of raster algorithms.

6.2.2.3 Uncompressed Mode

- For some detailed binary document images, both 1-D and 2-D RLC may result in data expansion instead of data compression.
- Under these circumstances the number of coding bits is larger than the number of bilivel pixels.
- An uncompressed mode is created as an alternative way to avoid data expansion. Special codewords are assigned for the uncompressed mode.
- For the performances of 1-D and 2-D RLC applied to eight CCITT test document images, and issues such as “fill bits” and “minimum scan line time (MSLT),” to only name a few, readers are referred to an excellent tutorial paper [hunter 1980].

6.3 Digital Facsimile Coding Standards

- Facsimile transmission, an important means of communication in modern society, is often used as an example to demonstrate the mutual interaction between widely used applications and standardization activities.

  - ♦ Active facsimile applications and the market brought on the necessity for international standardization in order to facilitate interoperability between facsimile machines worldwide.

  - ♦ Successful international standardization, in turn, has stimulated wider use of facsimile transmission and, hence, a more demanding market. Facsimile has also been considered as a major application for binary image compression.

Table 6.3 Facsimile coding standards

Group of facsimile apparatuses	Speed requirement for A-4 size document	Analog or digital scheme	CCITT recommen-dation	Compression Technique		
				Model	Basic coder	Algorithm acronym
G_1	6 min	Analog	T.2	—	—	—
G_2	3 min	Analog	T.3	—	—	—
G_3	1 min	Digital	T.4	1-D RLC 2-D RLC (optional)	Modified Huffman	MH MR
G_4	1 min	Digital	T.6	2-D RLC	Modified Huffman	MMR

6.4 Dictionary Coding

- Dictionary coding is different from Huffman coding and arithmetic coding.
    - ♦ Both Huffman and arithmetic coding techniques are based on a statistical model (e.g., occurrence probabilities).

- In dictionary-based data compression techniques, **a symbol or a string** of symbols generated from a source alphabet is represented by an **index** to a dictionary constructed from the source alphabet.

- A dictionary is a list of symbols and strings of symbols.
  - There are many examples of this in our daily lives.
    - the string “September” vs. “9,”
    - a social security number vs. a person in the U.S.
- Dictionary coding is widely used in text coding.
- Consider English text coding.
  - The source alphabet includes 26 English letters in both lower and upper cases, numbers, various punctuation marks and the space bar.

- ♦ Huffman or arithmetic coding treats each symbol based on its occurrence probability. That is, the source is modeled as a memoryless source.
- ♦ It is well known, however, that this is not true in many applications.
- ♦ In text coding, ***structure*** or ***context*** plays a significant role.
  - ▪ Very likely that the letter *u* appears after the letter *q*.
  - ▪ Likewise, it is likely that the word "concerned" will appear after "As far as the weather is."

- The strategy of the dictionary coding is to build a dictionary that contains frequently occurring symbols and string of symbols.
  - ♦ When a symbol or a string is encountered and it is contained in the dictionary, it is encoded with an index to the dictionary.

- ♦ Otherwise, if not in the dictionary, the symbol or the string of symbols is encoded in a less efficient manner.

- ❖ All the dictionary schemes have an equivalent statistical scheme, which achieves exactly the same compression.

- ❖ Statistical scheme using high-order context models may outperform dictionary coding (with high computational complexity).

- ❖ But, dictionary coding is now superior for fast speed and economy of memory.

- ❖ In future, however, statistical scheme may be preferred [bell 1990].

6.4.1 Formulation of Dictionary Coding

- Define dictionary coding in a precise manner [bell 1990].
  - We denote a source alphabet by S.

- A dictionary consisting of two elements is defined as $D=(P,C)$,
  - ✓ P is a finite set of phrases generated from the S,
  - ✓ C is a coding function mapping P onto a set of codewords.

- The set P is said to be complete if any input string can be represented by a series of phrases chosen from the P.
- The coding function C is said to obey the prefix property if there is no codeword that is a prefix of any other codeword.
- For practical usage, i.e., for reversible compression of any input text, the phrase set P must be complete and the coding function C must satisfy the prefix property.

6.4.2 Categorization of Dictionary-Based Coding Techniques

- The heart of dictionary coding is the formulation of the dictionary.

- A successfully built dictionary results in data compression; the opposite case may lead to data expansion.
- According to the ways in which dictionaries are constructed, dictionary coding techniques can be classified as static or adaptive.

6.4.2.1 Static Dictionary Coding

- A fixed dictionary,
  - Produced before the coding process
  - Used at both the transmitting and receiving ends
  - It is possible when the knowledge about the source alphabet and the related strings of symbols, also known as phrases, is sufficient.
- Merit of the static approach: its simplicity.
- Its drawbacks lie on
  - Relatively lower coding efficiency

- Less flexibility compared with adaptive dictionary techniques

♦ An example of static algorithms occurs is ***digram*** coding.

- A simple and fast coding technique.
- The dictionary contains:
    - ✓all source symbols and
    - ✓some frequently used pairs of symbols.
- In encoding, two symbols are checked at once to see if they are in the dictionary.
    - ✓If so, they are replaced by the index of the two symbols in the dictionary, and the next pair of symbols is encoded in the next step.
    - ✓If not, then the index of the first symbol is used to encode the first symbol. The second symbol is combined with the third symbol to form a new pair, which is encoded in the next step.

- ♦ The digram can be straightforwardly extended to ***n-gram***. In the extension, the size of the dictionary increases and so is its coding efficiency.

6.4.2.2 Adaptive Dictionary Coding

- A completely defined dictionary does not exist prior to the encoding process and the dictionary is not fixed.
  - At the beginning of coding, only an initial dictionary exists.
  - It adapts itself to the input during the coding process.
- All adaptive dictionary coding algorithms can be traced back to two different original works by Ziv and Lempel [ziv 1977, ziv 1978].
  - The algorithms based on [ziv 1977] are referred to as the LZ77 algorithms.

  - Those based on [ziv 1978] the LZ78 algorithms.

6.4.3 Parsing Strategy

- Once have a dictionary,
  - Need to examine the input text and find a string of symbols that matches an item in the dictionary.
  - Then the index of the item to the dictionary is encoded.
- This process of **segmenting the input text into disjoint strings** (whose union equals the input text) for coding is referred to as ***parsing***.
- Obviously, the way to segment the input text into strings is not unique.
- In terms of the highest coding efficiency, **optimal parsing** is essentially a <u>shortest-path</u> problem [bell 1990].

- In practice, however, a method called ***greedy*** **parsing** is used in all the LZ77 and LZ78 algorithms.
  - With greedy parsing, the encoder searches for the longest string of symbols in the input that matches an item in the dictionary at each coding step.
  - Greedy parsing may not be optimal, but it is simple in implementation.

Example 6.1

✓Consider a dictionary, D, whose phrase set $P = \{a, b, ab, ba, bb, aab, bbb\}$. The codewords assigned to these strings are $C(a) = 10$, $C(b) = 011$, $C(ab) = 010$, $C(ba) = 0101$, $C(bb) = 01$, $C(aab) = 11$, and $C(bbb) = 0110$.

✓Now the input text: *abbaab*.

✓Using greedy parsing, we then encode the text as $C(ab).C(ba).C(ab)$, which is a 10-bit string: 010.0101.010. In above representations, the periods are used to indicate the division of segments in the parsing.

✓ This, however, is not an optimum solution. Obviously, the following parsing will be more efficient, i.e., $C(a).C(bb).C(aab)$, which is a 6- bit string: 10.01.11.

6.4.4 Sliding Window (LZ77) Algorithms

6.4.4.1 Introduction

- The dictionary used is actually a portion of the input text, which has been recently encoded.
- The text that needs to be encoded is compared with the strings of symbols in the dictionary.
- The longest matched string in the dictionary is characterized by a *pointer* (sometimes called a *token*), which is represented by a triple of data items.
- Note that this triple functions as an index to the dictionary.

- In this way, a variable-length string of symbols is mapped to a fixed-length pointer.

- There is a sliding window in the LZ77 algorithms. The window consists of two parts: a search buffer and a look-ahead buffer.
  - The search buffer contains: the portion of the text stream that has recently been encoded --- the dictionary.
  - The look-ahead buffer contains: the text to be encoded next.

- The window slides through the input text stream from beginning to end during the entire encoding process.

- The size of the search buffer is much larger than that of the look-ahead buffer.
  - The sliding window: usually on the order of a few thousand symbols.
  - The look-ahead buffer: on the order of several tens to one hundred symbols.

6.4.4.2 Encoding and Decoding

Example 6.2

Encoding:

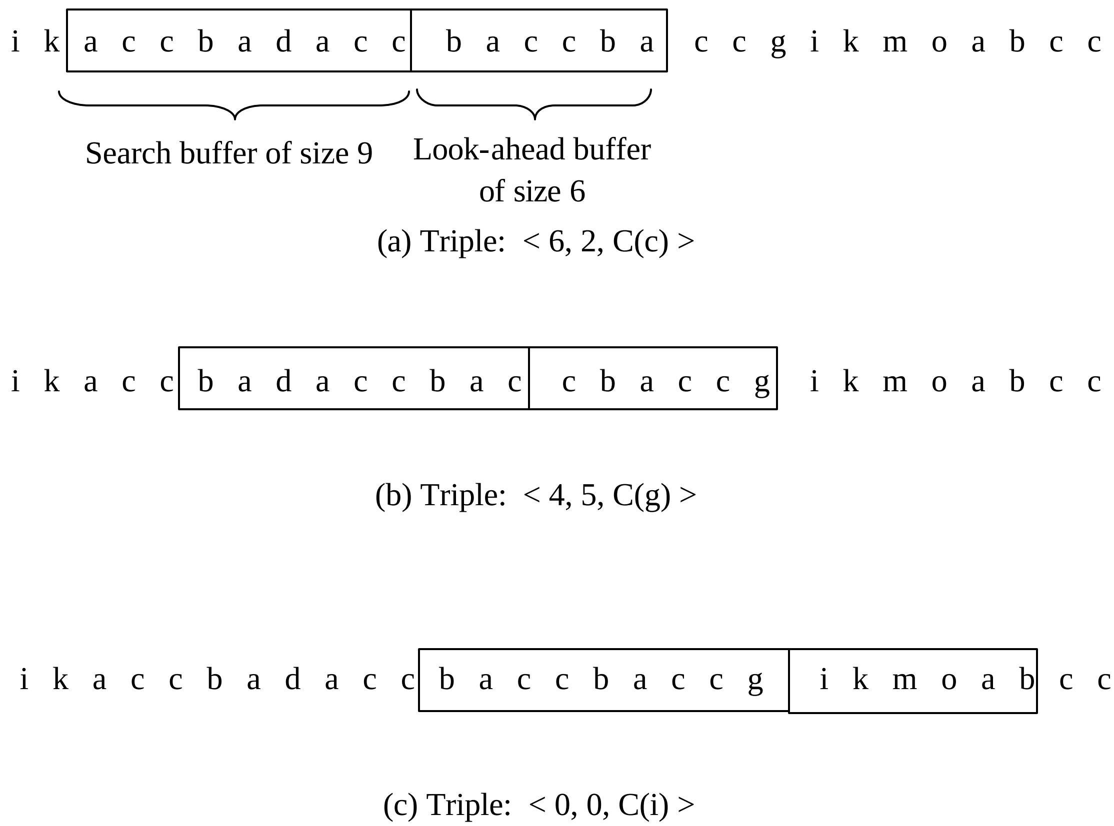


Figure 6.4 An encoding example using LZ77

Decoding:

✓Now let us see how the decoder recovers the string *baccbaccgi* from these three triples.

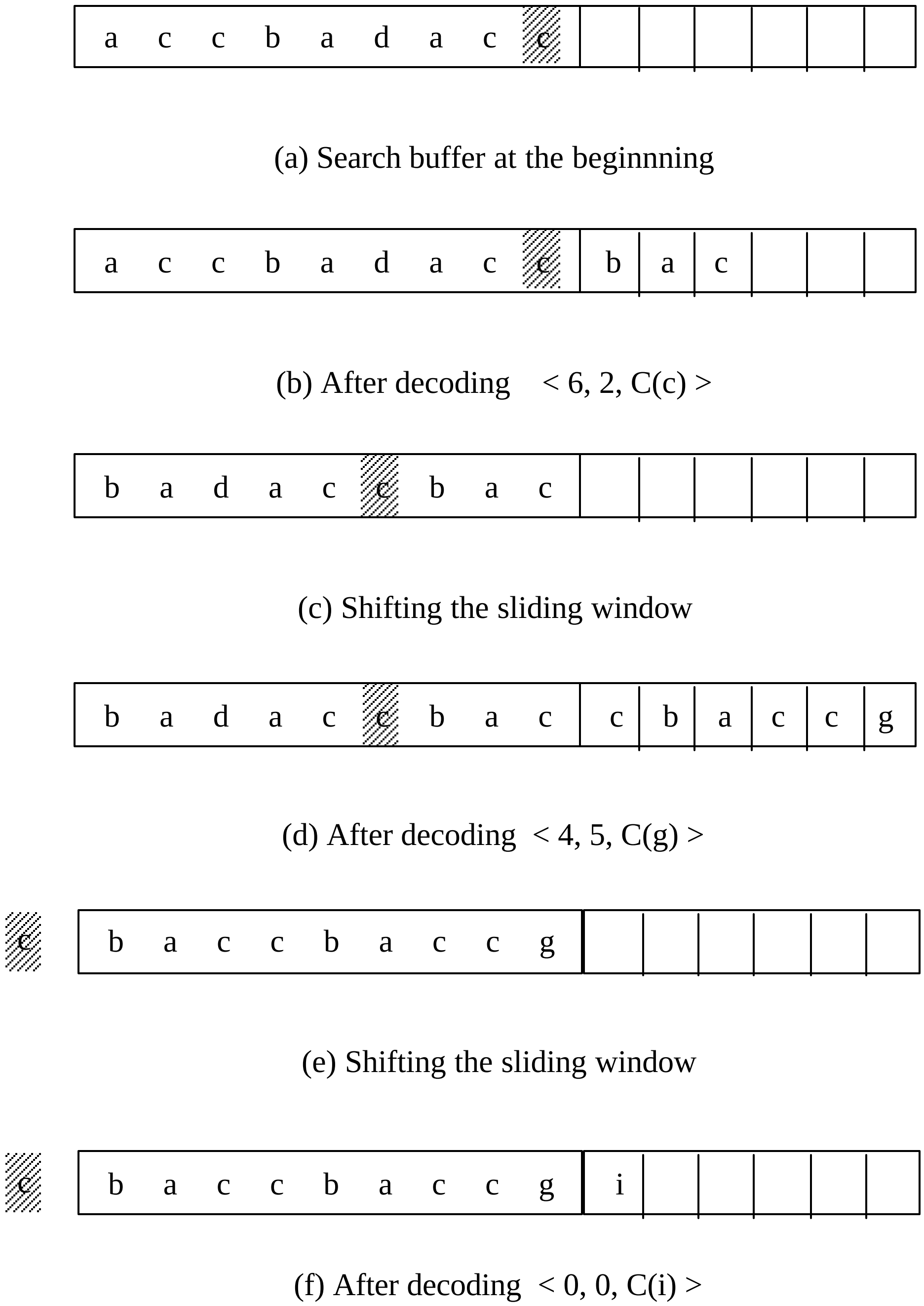


Figure 6.5 A decoding example using LZ77

✓In Figure 6.5, for each part, the last previously encoded symbol *c* prior to the receiving of the three triples is shaded.

6.4.4.3 Summary of the LZ77 Approach

➢The first symbol in the look-ahead buffer is the symbol or the beginning of a string of symbols to be encoded at the moment. Let us call it the symbol *s*.

➢In encoding, the search pointer moves to the left, away from the symbol *s*, to find a match of the symbol *s* in the search buffer. When there are multiple matches, the match that produces the longest matched string is chosen. The match is denoted by a triple <i, j, k>.

- The first item in the triple, “i”, is the **offset**, which is the distance between the pointer pointing to the symbol giving the maximum match and the symbol *s*.

- The second item, “j”, is the **length** of the matched string.
- The third item, “k”, is the codeword of the symbol following the matched string in the look-ahead buffer.

➢ At the very beginning, the content of the search buffer can be arbitrarily selected. For instance, the symbols in the search buffer may all be the space symbol.

♦ Denote the size of the search buffer by SB, the size of the look-ahead buffer by L and the size of the source alphabet by A. Assume that the natural binary code is used. Then we see that the LZ77 approach encodes variable-length strings of symbols with fixed-length codewords.

- Specifically, the offset “i” is of coding length $\lceil \log_2 SB \rceil$,
    - the length of matched string “j” is of coding length $\lceil \log_2 (SB + L) \rceil$,
    - and the codeword “k” is of coding length $\lceil \log_2 (A) \rceil$, where the sign $\lceil a \rceil$

denotes the smallest integer larger than a.

- The length of the matched string is equal to $\lceil \log_2 (SB + L) \rceil$ because the search for the maximum matching can enter into the look-ahead buffer.

- The decoding process is simpler than the encoding process since there is no comparison involved in the decoding.

- The most recently encoded symbols in the search buffer serve as the dictionary used in the LZ77 approach. The merit of doing so is that the dictionary is adapted to the input text well.

- The limitation of the approach is that if the distance between the repeated patterns in the input text stream is larger than the size of the search buffer, then the approach cannot utilize the structure to compress the text.

- A window with a moderate size, say, $SB + L \leq 8192$, can compress a variety of texts well. Several reasons have been analyzed in [bell 1990].

- Many variations have been made to improve coding efficiency of the LZ77 approach.

6.4.5 LZ78 Algorithms

6.4.5.1 Introduction

Limitations with LZ77:

- If the distance between two repeated patterns is larger than the size of the search buffer, the LZ77 algorithms cannot work efficiently.
- The fixed size of the both buffers implies that the matched string cannot be longer than the sum of the sizes of the two buffers, meaning another limitation on coding efficiency.
- Increasing the sizes of the search buffer and the look-ahead buffer seemingly will resolve the problems. A close look, however, reveals that it also leads to increases in the number of bits required to encode the offset and matched string

length as well as an increase in processing complexity.

LZ78:

- No use of the sliding window.
- Use encoded text as a dictionary which, potentially, does not have a fixed size.
- Each time a pointer (token) is issued, the encoded string is included in the dictionary.
- Once a preset limit to the dictionary size has been reached, either the dictionary is fixed for the future (if the coding efficiency is good), or it is reset to zero, i.e., it must be restarted.
- Instead of the triples used in the LZ77, only pairs are used in the LZ78.

  Specifically, only the position of the pointer to the matched string and the symbol following the matched string need to be encoded.

6.4.5.2 Encoding and Decoding

Example 6.3

Encoding:

- Consider the text stream: *baccbaccacbcabccbbacc*.
- In Table 6.4, in general, as the encoding proceeds, the entries in the dictionary become longer and longer. First, entries with single symbols come out, but later, more and more entries with two symbols show up. After that more and more entries with three symbols appear. This means that coding efficiency is increasing.

Table 6.4 An encoding example using the LZ78 algorithm

Index	Doubles	Encoded symbols
1	< 0, C(b) >	b
2	< 0, C(a) >	a
3	< 0, C(c) >	c
4	< 3, 1 >	cb
5	< 2, 3 >	ac
6	< 3, 2 >	ca
7	< 4, 3 >	cbc
8	< 2, 1 >	ab
9	< 3, 3 >	cc
10	< 1, 1 >	bb
11	< 5, 3 >	acc

Decoding:

Since the decoder knows the rule applied in the encoding, it can reconstruct the dictionary and decode the input text stream from the received doubles.

6.4.5.3 LZW Algorithm

- Both the LZ77 and LZ78 approaches, when published in 1977 and 1978, respectively, were theory-oriented.
- The effective and practical improvement over the LZ78 in [welch 1984] brought much attention to the LZ dictionary coding techniques. The resulting algorithm is referred to the LZW algorithm.
- It removed the second item in the double (the index of the symbol following the longest matched string) and hence, it enhanced coding efficiency.

  In other words, the LZW only sends the indexes of the dictionary to the decoder.

Example 6.4

Consider the following input text stream: *accbadaccbaccbacc*. We see that the source alphabet is $S=\{a,b,c,d,\}$.

Encoding:

Table 6.5 An example of the dictionary coding using the LZW algorithm

Index	Entry	Input Smbols	Encoded Index
1 2 3 4	a b c d	Initial dictionary	
5	ac	a	1
6	cc	c	3
7	cb	c	3
8	ba	b	2
9	ad	a	1
10	da	d	4
11	acc	a,c	5
12	cba	c,b	7
13	accb	a,c,c	11
14	bac	b,a,	8
15	cc…	c,c,…	

Decoding:

- Initially, the decoder has the same dictionary (the top four rows in Table 6.5) as that in the encoder.
- Once the first index 1 comes, the decoder decodes a symbol a.
- The second index is 3, which indicates that the next symbol is c.
  - ✓ From the rule applied in encoding, the decoder knows further that a new entry ac has been added to the dictionary with an index 5.
- The next index is 3. It is known that the next symbol is also c.
  - ✓ It is also known that the string cc has been added into the dictionary as the sixth entry.
- In this way, the decoder reconstructs the dictionary and decodes the input text stream.

6.4.5.4 Applications

- ♦ The CCITT Recommendation V.42 bis is a data compression standard used in modems that connect computers with remote users via the GSTN. In the compressed mode, the LZW algorithm is recommended for data compression.
- ♦ In image compression, the LZW finds its application as well. Specifically, it is utilized in the graphic interchange format (GIF) which was created to encode graphical images. GIF is now also used to encode natural images, though it is not very efficient in this regard. For more information, readers are referred to [sayood 1996].
- ♦ The LZW algorithm is also used in the Unix Compress command.

6.5 International Standards for Lossless Still Image Compression

6.5.1 Lossless Bilevel Still Image Compression

6.5.1.1 Algorithms

As mentioned above, there are **four** different international standard algorithms falling into this category.

- **MH** (Modified Huffman coding): This algorithm is defined in CCITT Recommendation T.4 for facsimile coding. It uses the 1-D RLC technique followed by the “modified” Huffman coding technique.
- **MR** (Modified READ (Relative Element Address Designate) coding): Defined in CCITT Recommendation T.4 for facsimile coding. It uses the 2-D RLC technique followed by the “modified” Huffman coding technique.

- **MMR** (Modified Modified READ coding): Defined in CCITT Recommendation T.6. It is based on MR, but is modified to maximize compression.
- **JBIG** (Joint Bilevel Image experts Group coding): Defined in CCITT Recommendation T.82. It uses an adaptive 2-D coding model, followed by an adaptive arithmetic coding technique.

6.5.1.2 Performance Comparison

- The JBIG test image set was used to compare the performance of the above-mentioned algorithms. The set contains scanned business documents with different densities, graphic images, digital halftones, and mixed (document and halftone) images.
- Note that **digital halftones**, also named (digital) halftone images, are generated by using only binary devices.
    - Some small black units are imposed on a white background.

- The units may assume different shapes: a circle, a square and so on.
- The more dense the black units in a spot of an image, the darker the spot appears.
- The digital halftoning method has been used for printing gray-level images in newspapers and books.

♦ For bilevel images excluding digital halftones, the compression ratio achieved by these techniques ranges from 3 to 100.

The compression ratio increases monotonically in the order of the following standard algorithms: MH, MR, MMR, JBIG.

♦ For digital halftones, MH, MR, and MMR result in data expansion, while JBIG achieves compression ratios in the range of 5 to 20.

This demonstrates that among the techniques, JBIG is the only one suitable for the compression of digital halftones.

6.5.2 Lossless Multilevel Still Image Compression

6.5.2.1 Algorithms

There are two international standards for multilevel still image compression:

- ❖ JBIG (Joint Bilevel Image experts Group coding): Defined in CCITT Recommendation T. 82.
    - It uses an adaptive arithmetic coding technique.
    - To encode multilevel images, the JIBG <u>decomposes multilevel images into bit-planes</u>, then compresses these bit-planes using its bilevel image compression technique.
    - To further enhance compression ratio, it uses Gary coding to represent pixel amplitudes instead of weighted binary coding.

- ❖ JPEG (Joint Photographic (image) Experts Group coding): Defined in CCITT Recommendation T. 81.

  For lossless coding (**Lossless JPEG**), it uses differential coding technique. The predictive error is encoded using either Huffman coding or adaptive arithmetic coding techniques.

6.5.2.2 Performance Comparison

- A set of color test images from the JPEG standards committee was used for performance comparison.
  - The luminance component (Y) is of resolution 720×576 pixels, while the chrominance components (U and V) are of 360×576 pixels.
  - The compression ratios calculated are the combined results for all the three components. The following observations have been reported.

- ♦ When quantized in 8 bits/pixel, the compression ratios vary much less for multilevel images than for bilevel images, and are roughly equal to 2.

- When quantized with 5 bits/pixel down to 2 bits/pixel, compared with the lossless JPEG, the JBIG achieves an increasingly higher compression ratio, up to a maximum of 29%.
- When quantized with 6 bits/pixel, JBIG and lossless JPEG achieve similar compression ratios.
- When quantized with 7 bits/pixel to 8 bits/pixel, the lossless JPEG achieves a 2.4 % to 2.6 % higher compression ratio than JBIG.

6.6 References

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

[arps 1979] R. B. Arps, "Binary Image Compression", in *Image Transmission Techniques*, W. K. Pratt (Ed.), New York: Academic Press, 1979.

[arps 1994] R. B. Arps and T. K. Truong, "Comparison of international standards for lossless still image compression," *Proceedings of The IEEE*, vol. 82, no. 6, pp. 889-899, June 1994.

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

[gonzalez 1992] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Reading, MA: Addison Wesley, 1992.

[hunter 1980] R. Hunter and A. H. Robinson, “International digital facsimile coding standards,” *Proceedings of The IEEE,* vol. 68, no. 7, pp. 854-867, 1980.

[laemmel 1951] A. E. Laemmel, “Coding processes for bandwidth reduction in picture transmission,” Rep. R-246-51, PIB-187, Microwave Res. Inst., Polytechnic Institute of Brooklyn, New York.

[nelson 1995] M. Nelson and J.-L. Gailly, *The Data Compression Book*, 2nd Edition, New York: M&T Books, 1995.

[sayood 1996] K. Sayood, *Introduction to Data Compression*, San Francisco, CA: Morgan Kaufmann Publishers, 1996.

[shannon 1949] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, Urbana, IL: University of Illinois Press, 1949.

[welch 1084] T. Welch, “A technique for high-performance data compression,” *IEEE Computer*, vol. 17, no. 6, pp. 8-19, June 1984.

[ziv 1977] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on Information Theory,* vol. 23, no. 3, pp. 337-343, May 1977.

[ziv 1978] J. Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding,” *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530-536, Sep. 1978.