Text compression with an Associative Parallel Processor

R. M Lea

Department of Electrical Engineering and Electronics, Brunel University, Kingston Lane, Uxbridge, Middlesex UB8 3PH

Text compression, using a coding dictionary of 200+ n-grams, can halve file storage costs and double data transmission rates. However, software based text compression systems are slow and expensive in storage.

Two hardware systems (based on a fixed record length and a byte-organised variable record length Associative Parallel Processor), for the compression and decompression of textual information, are described. Algorithms are given and their execution illustrated with practical examples.

A feasibility study, comparing the performances and costs of the two systems with a conventional microprocessor (Digital LSI-11) implementation is also reported. The fixed record length system is 3,600 times faster for compression, 1,600 times faster for decompression and its production cost is nearly 6% cheaper. The variable record length system is 560 times faster for compression, 240 times faster for decompression and its production cost is 58% cheaper. Whereas the conventional micro-processor system cannot perform at typical disk transmission rates, the compression and decom-pression rates for the fixed record length hardware are 4.1 and 6.3M bytes-per sec. and for the variable record length hardware are 0.64 and 0.91M bytes-per-sec. respectively.

(Received May 1976)

1. Introduction

On-line, large-scale, information retrieval systems require direct access to large (viz. hundreds of M bytes) data-bases. Moreover, bibliographic text files are updated at regular intervals. For example, one year of 'Chemical Abstracts Condensates' contains about 4 M words. Such large data-bases impose high storage and maintenance costs. In addition there is an increasing demand for small-scale information retrieval systems in the new growth areas of office and library automation. In view of the 'information explosion', it is essential that the storage requirement of retrieval systems is minimised or, put another way, that the information content of a storage medium is maximised. Thus, the subject of text compression is becoming increasingly important.

Text compression is achieved by reducing the intrinsic redundancy of stored text. Assuming the removal of unneces-sary spaces or other delimiters, considerable redundancy may still exist in the encoding of the information content of text files. Most computer systems employ a standard 8-bit character code (e.g. ASCII and EBCDIC), which can support 256 different code symbols. However, many text files make use of only upper-case characters and a few special symbols for punctuation and system control. A typical allocation of code symbols is as follows:

Upper-case characters	26
Numerals	10
Special symbols	14
Unused symbols	206
TOTAL	256

Since more than 200 of the 256 code symbols are unused, this form of encoding is clearly wasteful of storage. In addition, text strings contain considerable redundancy due to the disparate frequency distribution of characters and words. Zipf (Fairthorne, 1969) has shown that if the different symbols (viz. characters or words) of a large text sample are ranked in order of decreasing frequency then the frequency-rank distribution is hyperbolic. This implies that 50% of the different words in a data base occur only once and less than 17% of the different words occur more than twice (Booth, 1967). The hyper-bolic distribution of symbols in textual files has been verified by the work of many investigators (Clare, Cook and Lynch, 1972; Heaps, 1972; Lynch, Petrie and Snell, 1973; Schuegraf and Heaps, 1973). A typical distribution is shown in **Table 1**.

Shannon (1948; 1951) measured the amount of information associated with a symbol-set of N different symbols in terms of 'entropy' (viz. degree of uncertainty). The average entropy, H, for a textual file is given by the expression.

$$\text{Average entropy} = H = -\sum_{i=1}^{N} p_i \log_2 p_i \text{ bits per symbol} \quad (1)$$

where p_i is the probability of occurrence of the ith symbol within the file.

If it were possible to code text symbols with a non-integer number of physical bits, then the average entropy associated with a symbol-set could be regarded as the minimum code-length required. Thus, the minimum storage (S_m) requirement for a textual file comprising a total of T symbols, is given by

$$\text{Minimum storage requirement} = S_m = H.T \text{ bits} \quad (2)$$

The average entropy (H) associated with a symbol-set also provides a measure of the efficiency (or degree of inherent redundancy) with which the symbol-set supports the stored information. Evaluation of equation (1) shows that H reaches a maximum value (H_M) when all symbols are equally frequent.

Table 1 Frequency distribution of characters in the title field of the INSPEC data-base ranked in inverse frequency order (Lynch, 1975).

Rank	*Character*	*Frequency*
1	Space	11,003
2	E	7,497
3	I	6,117
4	T	5,865
23	W	430
24	X	236
25	K	159
26	Q	149
50	%	3
51	;	2
52	=	1
53	£	1

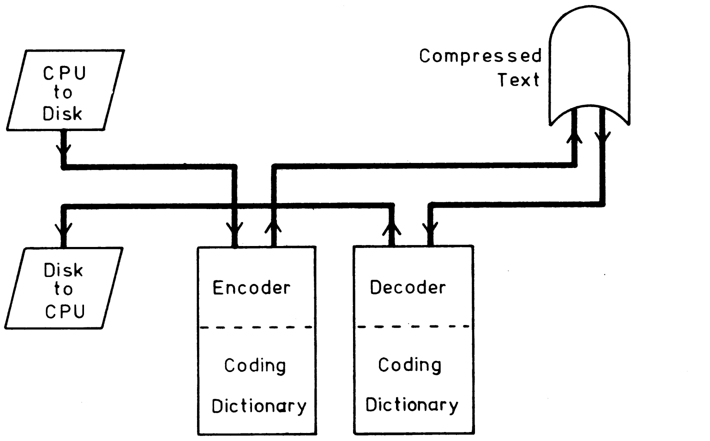


Fig. 1 Schematic organisation of a text compressor/decompressor module

For equifrequency

$$f_i = \frac{T}{N} \text{ for all } i \qquad (3)$$

where f_i is the frequency of occurrence of the ith symbol. Hence, for equifrequency

$$P_i = \frac{f_i}{T} = \frac{1}{N} \qquad (4)$$

Substituting equations (3) and (4) in equation (1)

$$H_M = \log_2 N \text{ bits per symbol} \qquad (5)$$

The maximum value (H_M) of the average entropy (H) of a symbol-set is a measure of the maximum amount of information which the symbol-set can represent. Hence, the efficiency of information representation depends on the degree of equifrequency (E) (also known as 'relative entropy') which the symbol-set attains, as defined by the expression

$$\text{Degree of equifrequency} = E = \frac{H}{H_M} \qquad (6)$$

The degree of equifrequency, E, measures the uniformity of the symbol frequency distribution and has unity value for a uniform distribution. Hence E provides the measure of coding efficiency, which is required to analyse the information content of text files, since if the file is optimally coded it must contain information in its most compact form. The value of E for the characters and words of typical bibliographic data-base is 0·75.

Many text compression techniques have been investigated, with the objective of reducing the intrinsic redundancy of text storage. These involve more efficient encoding of characters, words and word and text fragments (variable-length character strings). Consequently a text compressor/decompressor module comprises an encoder in the input channel to the storage device and a decoder in the output channel. Both the encoder and decoder sub-modules require rapid access to a coding dictionary. A schematic organisation for a text compressor/decompressor module is shown in **Fig. 1.**

Character encoding
Shannon (1948; 1951), Huffman (1952) and Fano (1961) independently devised text compression schemes which involve the assignment of variable-length binary codes to each character; the code-length being inversely proportional to the probability of occurrence of the character. Thus the most frequent characters are assigned the shortest codes (e.g. a 4 bit code for 'E') and conversely, longer codes are assigned to the less frequent characters (e.g. a 12 bit code for 'K'). Such codes achieve a near unity value for the degree of equifrequency and a compression ratio (viz. compressed text: natural text) of just over 50%. The coding dictionary would require the 50-odd entries corresponding to the basic character-set of the data-base.

Word encoding
Ruth and Kreutzer (1972) and Wells (1972) have considered the use of Huffman codes (1952) for word encoding. Moreover Gilbert and Moore (1959) have shown that this technique provides the most compact coding for words. To minimise the inconvenience of handling variable-length bit-strings, Thiel and Heaps (1972) restricted code-lengths to an integer number of bytes. A program for coding was reported (Heaps and Thiel, 1970) and analysis of the storage requirements for such compression coding has been performed (Heaps, 1972). Such techniques achieve near unity values for the degree of equifrequency and a compression ratio of just over 16%. Unfortunately, word-encoding requires a large coding dictionary which is highly data-base dependent and grows with each update. Hence encoding and decoding are slow in operation and expensive in storage requirement.

Fragment encoding
In contrast to character and word encoding schemes, in which basic language elements are assigned variable-length binary codes, fragment encoding involves the assignment of fixed-length binary codes to variable-length character strings. Many researchers, notably Lynch (Clare, Cook and Lynch, 1972; Lynch, Petrie and Snell, 1973; Fokker and Lynch, 1974; 1975). Schuegraf and Heaps (1973; 1974), have investigated new symbol sets comprising word or text fragments (often called n-grams) which are stored and transmitted as fixed-length codes. For example, an 8-bit byte can support 256 fragment codes, including the basic character set, as shown in **Table 2.** Fragment encoding achieves values for the degree of equifrequency in the range of 0·95–0·98 and compression ratios in the range of 30%–70% depending on the size of the coding dictionary, which is in the range of 100–2,000 entries.

Most text compression investigations have been mainly concerned with the creation of the coding dictionary. All reported text compressor/decompressor modules have been implemented entirely in software. Few modules have been used in commercial retrieval systems, although some evaluation studies have been undertaken using samples of commercial data bases. The degrees of compression, claimed by investigators, vary from 30% to 84% depending on the encoding technique and the size of the dictionary.

The advantages of text compression are two-fold:

1. Reduction of 30%–84% in file storage requirement. Hence, for files held on disk storage:
   (*a*) more records can be stored per track (of particular advantage for cellular files) (Dodd, 1969; Lefkovitz, 1969)
   (*b*) inter-record distances can be reduced (of particular advantage for linked and inverted files (Dodd, 1969; Lefkovitz, 1969)) because head movements per record retrieval are reduced.
2. Increase of 43%–625% in text transmission speeds when reading from the disk.

The above advantages are offset by the complexity of the software implementation of the text compressor/decompressor module, which incurs overheads in execution time and program storage. These disadvantages seriously detract from the advantages of text compression.

This paper considers the implementation of a text compressor/decompressor in hardware with the objective of achieving a

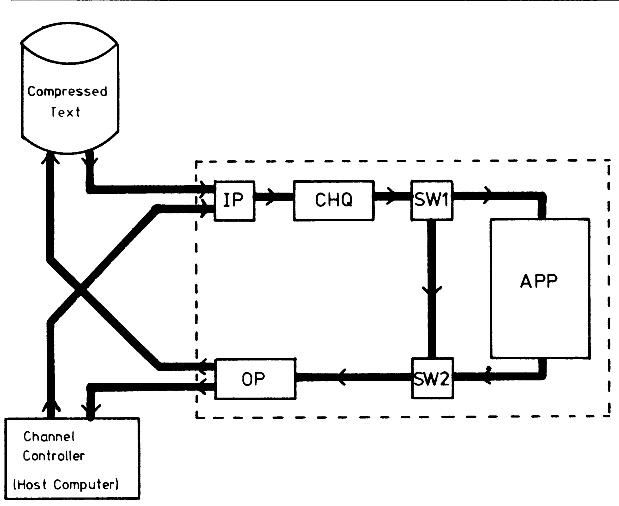
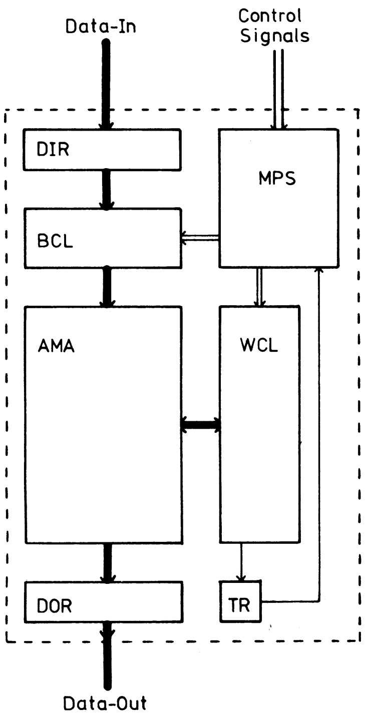


Fig. 2 Schematic organisation of the text compression hardware
Legend

IP	= Input Port
OP	= Output Port
CHQ	= Character Queue
SW1/2	= Text routing switches
APP	= Associative Parallel Processor



Legend

MPS	= Micro-Program Store
DIR	= Data Input Register
DOR	= Data Output Register
AMA	= Associative Memory Array
BCL	= Bit Control Logic
WCL	= Word Control Logic
TR	= Tag Reply

Fig. 3 Organisation of the Associative Parallel Processor

Table 2 The 256 selected *n*-grams for two different bibliographic data-bases (Clare, Cook and Lynch, 1972; Lynch, 1975)

n	*Example*	*INSPEC titles*	*Chemical titles*
1	N	53	47
2	□O	128	136
3	ION	52	49
4	THE□	7	10
5	□AND□	8	6
6	ATION□	4	3
7	ION□OF□	2	2
8	□OF□THE□	2	2
9	ATION□OF□	—	1

□ = Space

reasonable degree of compression at a low cost and avoiding the execution time penalty of the reported software modules. A survey of text compression techniques resulted in the choice of an *n*-gram encoding scheme as reported by Lynch (Clare, Cook and Lynch, 1972; Lynch, Petrie and Snell, 1973; 1975), for the following reasons:

1. Convenient text manipulation, since the technique is based on 8-bit bytes.
2. 50% compression can be achieved with a fairly small coding dictionary (viz. approximately 200 entries).
3. The coding dictionary is insensitive to changes within a data-base or between similar data-bases.
4. The size of coding dictionary can be easily controlled.
5. The technique has been thoroughly investigated, results are consistent and the relevant expertese is easily available.

Using a coding dictionary supplied by Lynch and a copy of the INSPEC data-base a text compression vehicle has been established at Brunel University. Text compression/decompression modules, based on simulations of the Associative Parallel Processor (APP) and a conventional micro-processor, have been designed and tested. Further to these investigations this paper reports that the APP offers a low-cost high-speed alternative to conventional hardware and software routines for text compression and decompression tasks.

2. System description

The text compression hardware comprises a microprogrammed Associative Parallel Processor (APP), a Character Queue (CHQ), access ports and switches as shown in **Fig. 2.** The unit is incorporated in a DMA channel of the host computer, which handles all text transfers external to the compression hardware. A schematic diagram of the APP is shown in **Fig. 3** and a description of its operation is given below. The *n*-gram dictionary is stored in the associative memory of the APP as an arrangement of ordered pairs (viz. ⟨*n*-gram, code⟩). For example the content of the Associative Memory could form the set *S*, where,

$S = \{\langle \text{TION}, \alpha\rangle\langle \text{THE }, \beta\rangle\langle \text{AN}, \delta\rangle, \ldots, \langle \text{TI}, \gamma\rangle\langle \text{ OF }, \varepsilon\rangle\}$

If only the upper case character alphabet (50 characters) is used and the APP has an 8-bit character field then the *n*-gram dictionary has 206 entries. It can be seen from Table 2 that pentagrams and higher order *n*-grams may provide less than 6% of the new symbol set. Consequently tetragrams have been chosen for the largest *n*-gram supported by the text compression system described in this paper. During compression, the input text string is transferred character by character from the Input Port (IP) (see Fig. 2) to the Character Queue (CHQ). When the CHQ is fully loaded, characters are transferred to the Data Input Register (DIR) (see Fig. 3) of the APP, and a search of the associative memory is initiated. If an *n*-gram is located, the corresponding code is transferred, via the Data Output Register (DOR) and the switch SW2, to the Output Port (OP), which takes the form of a block buffer. In the event of a mismatch a single character is transferred from the CHQ, via the switches SW1 and SW2, to the OP. In either case, the appropriate number of new characters are loaded into the CHQ. Hence, during compression the APP recognises *n*-grams in the input text string and replaces them with their corresponding codes. For example, using the *n*-gram dictionary shown in **Fig. 4,** if the text string at the Input Port (IP) is . . . THE ACTION OF TIN . . . the resulting text string at the Output Port (OP) would be . . . βAC $\alpha\varepsilon\gamma$N . . . .

The system operates in reverse during text decompression. As *n*-gram codes are recognised they are replaced by the corresponding *n*-gram characters from the dictionary.

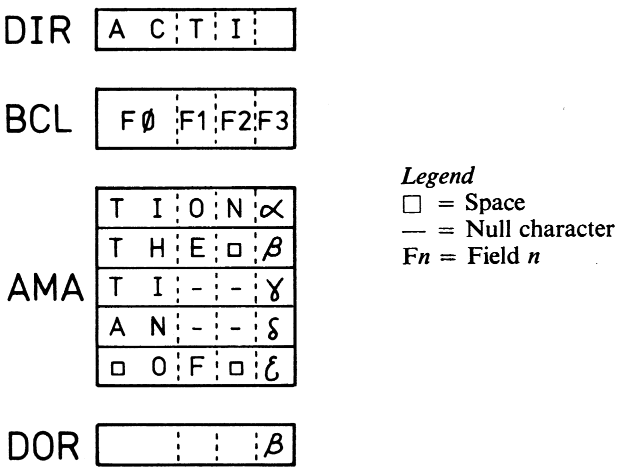


Fig. 4 **Storage allocation for the *n*-gram dictionary in a fixed record length APP**

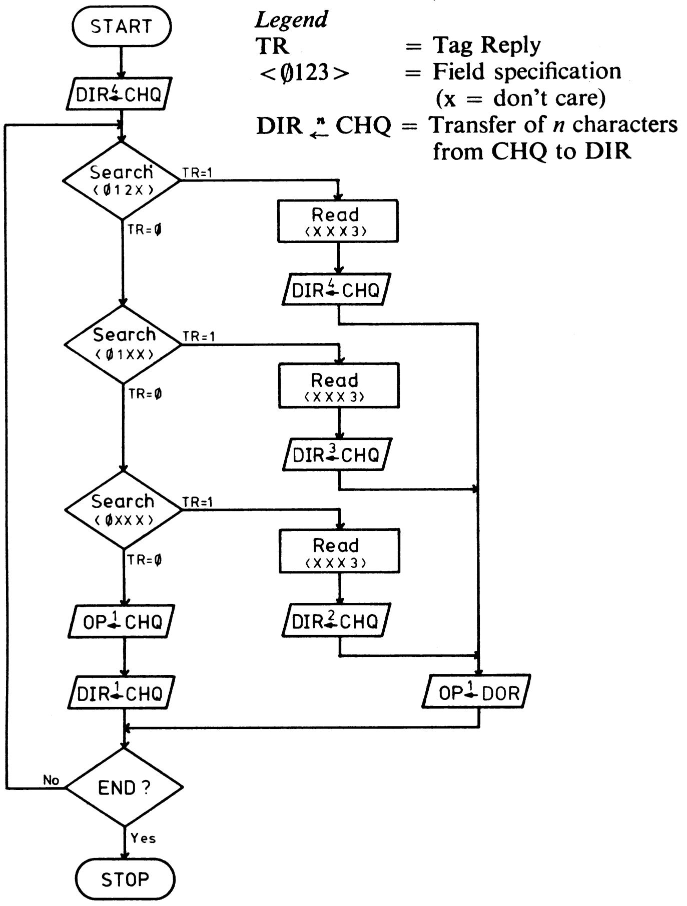


Fig. 5 **Algorithm for text compression with a fixed record length APP**

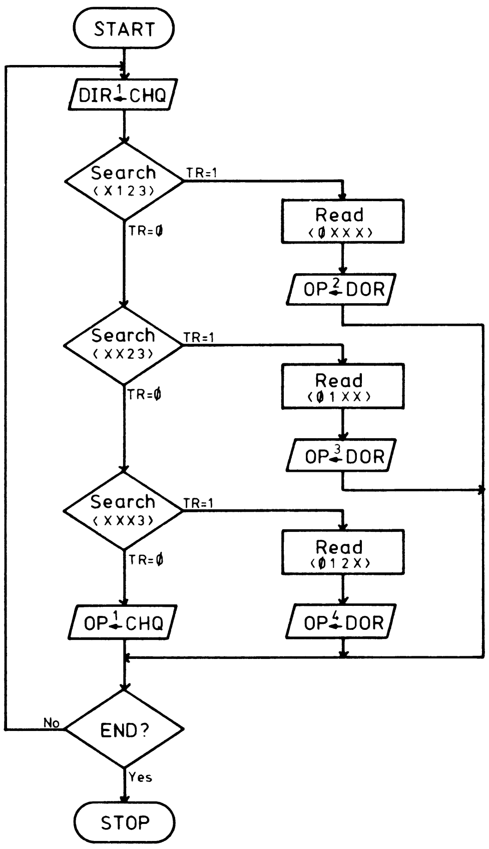


Fig. 6 **Algorithm for text decompression with a fixed record length APP**

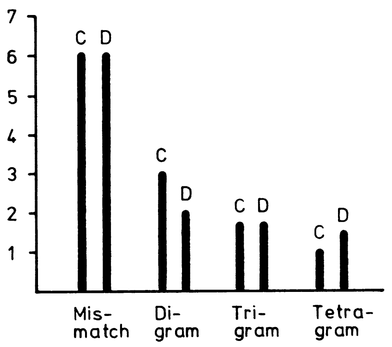


Fig. 7 **Instruction execution counts per character for text compression (C) and decompression (D) with a fixed record length APP**

2.1. *APP organisation*

The APP comprises an Associative Memory Array (AMA) (also known as a word organised content addressable memory), Data Input and Output Registers (DIR and DOR), and a Micro-Program Store (MPS), as shown in Fig. 3. The associative memory can be considered as a two dimensional array of identical one-bit processing elements, or cells. Each cell has storage for one bit, and contains sufficient logic to enable its content to be compared with the corresponding bit of the Data Input Register (DIR). All word-rows of the associative memory can be accessed in parallel by the Word Control Logic (WCL), and particular bit columns can be selected in parallel by the Bit-Control Logic (BCL). Three basic operations can be performed on the contents of the Associative Memory Array:

SEARCH: All the cells in each selected bit column simultaneously compare their contents with the corresponding bit of the DIR, and the appropriate match or mismatch outputs are recorded in a tag register in the WCL. The output for each word is decided as follows: the cells in all unselected bit columns give match outputs and these are logically ANDed with the output of each selected cell within the word-row. After the search operation, the contents of the tag register can be propagated to an adjacent word or 'run' to the end of the memory.

READ: All the cells in the 'tagged' word-row simultaneously transfer their contents to the DOR.

WRITE: All the cells in each selected bit-column of all tagged word-rows change their contents to that of the corresponding bits in the DIR.

The control signals governing the operation of the Bit and Word Control Logic units are obtained from the MPS. Branch instructions operate on a reply (TR) from the memory, which is

Table 3 Example of text compression with a fixed record length APP

Step	*Search criteria*	*Output character*
1	T H E □	β
2	A C T I	
3	A C T X	
4	A C X X	A
5	C T I O	
6	C T I X	
7	C T X X	C
8	T I O N	α
9	□O F □	ε
10	T I N□	
11	T I NX	
12	T I X X	γ
13	N□□□	
14	N□□X	
15	N□X X	N

□ = Space
Input text string: THE ACTION OF TIN
Output text string: βACαεγN

Table 4 Example of text decompression with a fixed record length APP

Step	*Search criteria*	*Output characters*
1	X——β	
2	X X—β	
3	X X Xβ	T H E □
4	X——A	
5	X X—A	
6	X X X A	A
7	X——C	
8	X X—C	
9	X X X C	C
10	X——α	
11	X X—α	
12	X X Xα	T I O N
13	X——ε	
14	X X—ε	
15	X X Xε	□O F □
16	X——γ	T I
17	X——N	
18	X X—N	
19	X X X N	N

— = Null character
□ = Space
Input text string: βACαεγN
Output text string: THE ACTION OF TIN

generated by a logical OR operation on the contents of the tag register. Further information on the structural organisation of the APP can be found in Lea (1975c).

3. System operation

The operation and performance of the text compression system, shown in **Fig. 2**, depends on the organisation of the *n*-gram dictionary within the Associative memory. There are two basic forms, namely the fixed record length and the byte-organised variable record length (Lea, 1975c).

3.1. *Fixed record length* n*-gram dictionary*

One word-row of the memory array is allocated to each ordered pair (viz. ⟨*n*-gram, code⟩) and the hardware supports Search-Read operations as primitive functions. If the dictionary is made up of digrams, trigrams and tetragrams, the Bit Control Logic of the APP is organised to support four fields, as shown in Fig. 4. The compression and decompression algorithms for this organisation are shown in **Figs. 5** and **6** and their performance criteria are summarised in **Fig. 7.** Practical examples for the fixed record length text compression hardware are given in **Tables 3** and **4.**

A disadvantage of the fixed record length organisation is the storage redundancy caused when the *n*-grams are of different lengths. The variable record length mode of organisation avoids this redundancy.

3.2. *Byte-organised variable record length* n*-gram dictionary*

One word-row in the memory array is allocated to one character such that each ordered pair, in this case ⟨code, *n*-gram⟩, is stored in a group of contiguous memory words; as shown in **Fig. 8.** A control-bit which is set in Field 1 marks the beginning of each *n*-gram and a control-bit which is set in Field 2 marks each *n*-gram code. The association links between the characters of each ordered pair are represented by the propagation of a control-bit in Field 3. The variable record length text compression hardware supports a microprogrammable instruction which executes all, or part of, the sequence 'Search—Clear (multi-Write ∅)—Propagate (Run and Resolve) Tags—Read—Write'. The compression and decompression algorithms for this organisation are shown in **Figs. 9** and **10** and their performance criteria are summarised in **Fig. 11.** Practical examples for the variable record length text compression hardware are given in **Tables 5** and **6.**

The variable record length organisation can be regarded as an efficient implementation of the cellular string memory originally proposed by Lee (Lee, 1962; Beaven and Lewin, 1972; Lea and Wright, 1973).

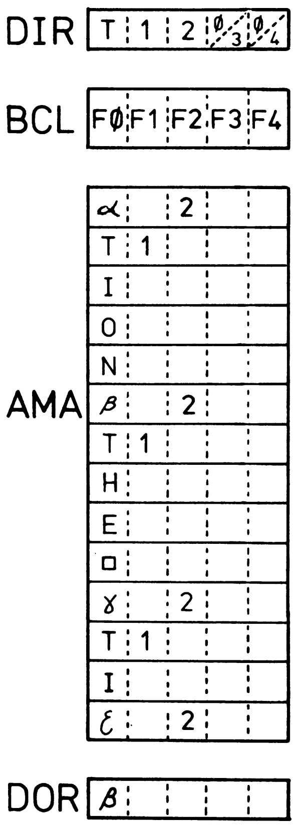


Fig. 8 Storage allocation for the *n*-gram dictionary in a variable record length APP

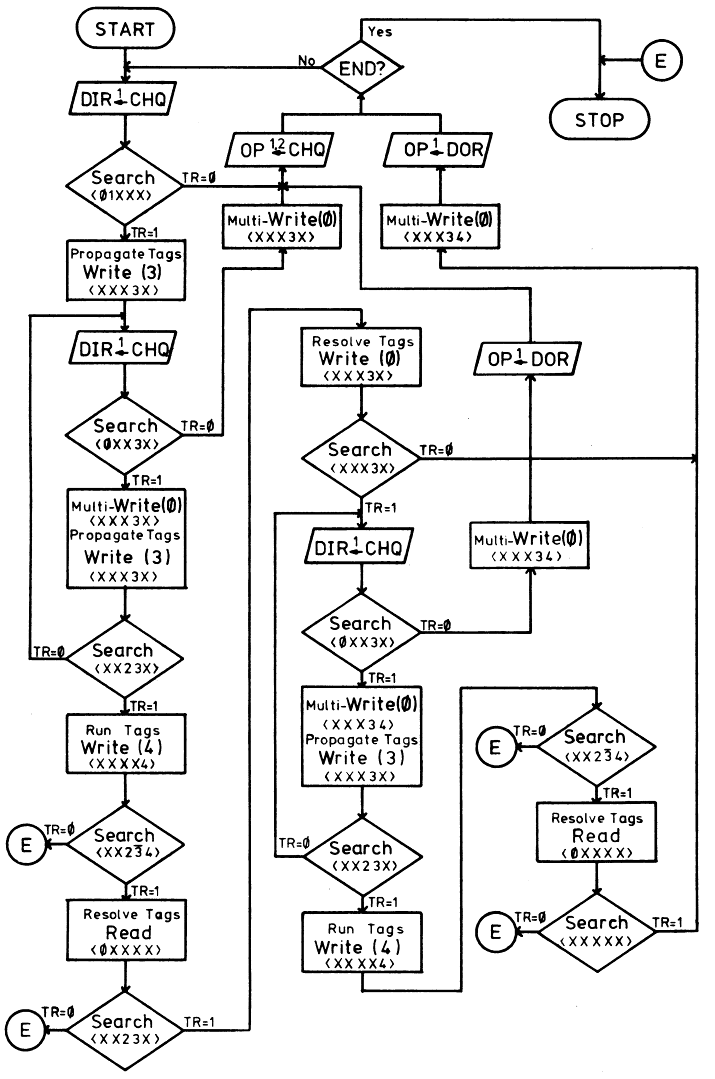


Fig. 9 **Algorithm for text compression with a byte-organised variable record length APP**

Legend

E = Error condition

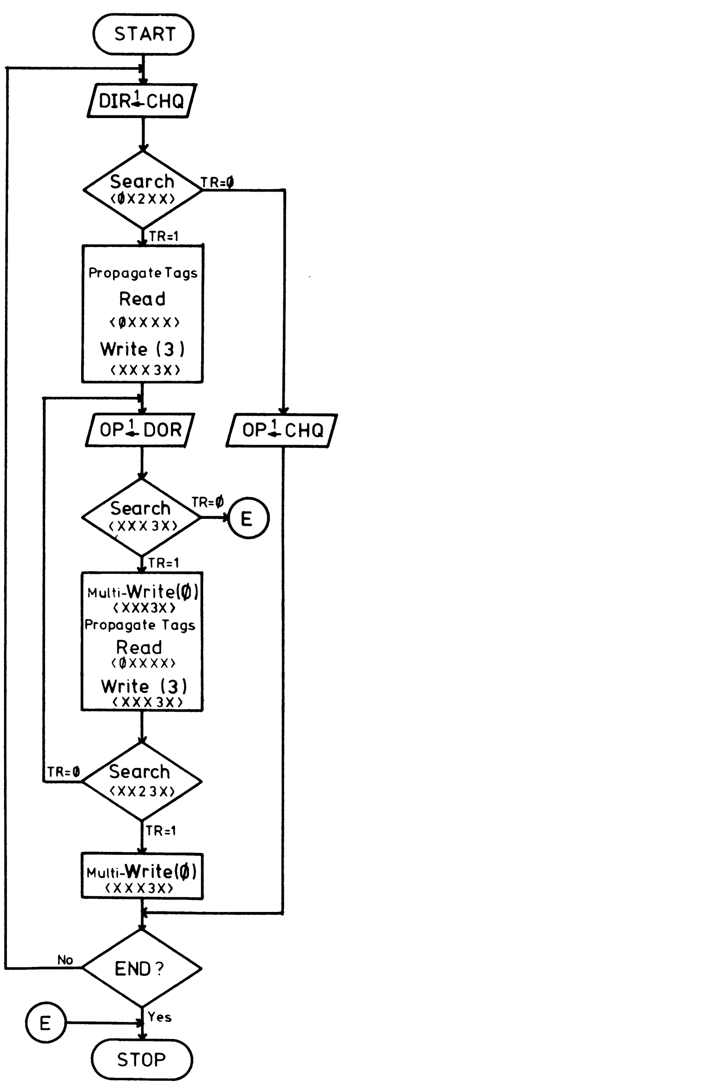
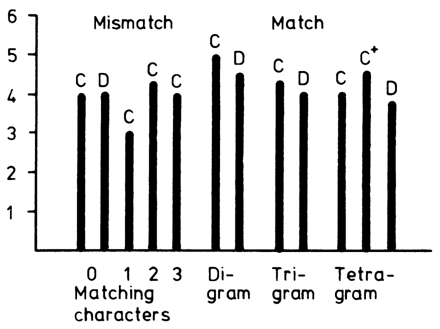


Fig. 10 **Algorithm for text decompression with a byte-organised variable record length APP**



Legend

+ = Including a digram

Fig. 11 Instruction execution counts per character for text compression (C) and decompression (D) with a byte-organised variable record length APP

4. System performance

A comparative study has been carried out at Brunel University to investigate the feasibility of the proposed system for text compression and decompression tasks (Donnelly and Lea, 1975; Donnelly and Mottershead, 1976; Donnelly, 1976a; 1976b). Three assembly language programs were developed for execution on a PDP 11/40 mini-computer system to provide three different representations of the *n*-gram dictionary; these were:

1. A conventional approach, using binary tree searching and table look-up routines, as suggested in Byrne and Mullarney (1972) and shown in the Appendix.
2. A simulation of the fixed record length system, shown in Figs. 2, 3, 4, 5, and 6.
3. A simulation of the variable record length system shown in Figs. 2, 3, 8, 9, and 10.

The three programs were applied to various compression and decompression tasks and their performances, in terms of instruction execution counts and *n*-gram dictionary storage requirements were compared. The results of this investigation are shown in **Table 7.**

5. System implementation

Despite numerous research investigations (Hanlon, 1966; Minker, 1971; Parhami, 1973) the economic manufacture of high speed associative memories has been regarded as impossible for many years. However, recent advances in LSI fabrication techniques can now be exploited to put an end to this view. Many designs for both bipolar and especially MOS Content Addressable Memory (CAM) devices have been proposed (Lea, 1975a; 1975b; 1976; 1977a; 1977b). Of these, two particular MOS CAM designs (Lea, 1975a; 1977a; 1977b) would be suitable for the manufacture of text compression hardware.

The fixed record length APP could be implemented with the

Table 5 Example of text compression with a byte-organised variable record length APP

Step	*Search criteria*	*Associative memory content*																			*DOR content*	*Output character*
		F4																				
		F3																				
		F2	2					2					2			2					2		
		F1		1					1					1			1						
		F0	α	T	I	O	N	β	T	H	E	□	γ	T	I	ε	□	O	F	□	δ		
1, 2	T	F3			3					3					3								
3, 4	H	F3									3												
5-7	E	F3										3											
8-10	□	F3											3										
11, 12		F4	4	4	4	4	4	4	4	4	4	4	4										
		F3											3									β	
13		F4	4	4	4	4	4	4	4	4	4	4	4										
		F3																				β	
14-16		F4																					
		F3																					
		F2	2					2					2			2					2		
		F1		1					1					1			1						
		F0	α	T	I	O	N	β	T	H	E	□	γ	T	I	ε	□	O	F	□	δ	β	β
17-20	A	F3																				β	A
21-24	C	F3																				β	C
25, 26	T	F3			3					3					3							β	
27, 28	I	F3				3										3						β	
29, 30		F4	4	4	4	4	4	4	4	4	4	4	4	4	4	4							
		F3				3										3						γ	
31		F4	4	4	4	4	4	4	4	4	4	4	4	4	4	4							
		F3				3																γ	
32-34	O	F4																					
		F3					3															γ	
35-37	N	F3						3														γ	
38, 39		F4	4	4	4	4	4	4															
		F3						3														α	
40-42		F4																					
		F3																					
		F2	2					2					2			2					2		
		F1		1					1					1			1						
		F0	α	T	I	O	N	β	T	H	E	□	γ	T	I	ε	□	O	F	□	δ	α	α
43, 44	□	F3																3				α	
45, 46	O	F3																	3			α	
47-49	F	F3																		3		α	
50-52	□	F3																			3	α	
53, 54		F4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4		
		F3																			3	ε	
55		F4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4		
		F3																				ε	
56-58		F4																					
		F3																					
		F2	2					2					2			2					2		
		F1		1					1					1			1						
		F0	α	T	I	O	N	β	T	H	E	□	γ	T	I	ε	□	O	F	□	δ	ε	ε
59, 60	T	F3			3					3					3							ε	
61, 62	I	F3				3										3						ε	
63, 64		F4	4	4	4	4	4	4	4	4	4	4	4	4	4	4							
		F3				3										3						γ	
65		F4	4	4	4	4	4	4	4	4	4	4	4	4	4	4							
		F3				3																γ	
66-69	N	F4																					
		F3																				γ	γ
70, 71		F3																				γ	N

□ = Space
Input text string: THE ACTION OF TIN
Output text string: βACαεγN

Table 6 Example of text decompression with a byte-organised variable record length APP

Step	*Search criteria*	*Associative memory content*																				*DOR content*	*Output character*
		F4																					
		F3																					
		F2	2					2					2			2					2		
		F1		1					1					1			1						
		F∅	α	T	I	O	N	β	T	H	E	□	γ	T	I	ε	□	O	F	□	δ		
1, 2	β	F3							3													T	
3,4		F3								3												H	T
5-7		F3									3											E	H
8-10		F3										3										□	E
11-13		F3											3									γ	□
14,15		F4																					
		F3																					
		F2	2					2					2			2					2		
		F1		1					1					1			1						
		F∅	α	T	I	O	N	β	T	H	E	□	γ	T	I	ε	□	O	F	□	δ	γ	
16-19	A	F3																				γ	A
20-23	C	F3																				γ	C
24, 25	α	F3		3																		T	
26, 27		F3			3																	I	T
28-30		F3				3																O	I
31-33		F3					3															N	O
34-36		F3						3														β	N
37, 38		F4																					
		F3																					
		F2	2					2					2			2					2		
		F1		1					1					1			1						
		F∅	α	T	I	O	N	β	T	H	E	□	γ	T	I	ε	□	O	F	□	δ	β	
39, 40	ε	F3															3					□	
41, 42		F3																3				O	□
43-45		F3																	3			F	O
46, 47		F3																		3		□	F
48-50		F3																			3	δ	□
51, 52		F4																					
		F3																					
		F2	2					2					2			2					2		
		F1		1					1					1			1						
		F∅	α	T	I	O	N	β	T	H	E	□	γ	T	I	ε	□	O	F	□	δ	δ	
53, 54	γ	F3												3								T	
55, 56		F3													3							I	T
57-59		F3														3						ε	I
60. 61		F3																				ε	
62-65	N	F3																				ε	N

□ = Space
Input text string: βACαεγN
Output text string: THE ACTION OF TIN

MOS CAM design described in Lea (1975a). This device would provide 16 words of 8-bits each in a 36-pin plastic dual-in-line pack. Hence 62 of these packs would be required to implement the *n*-gram dictionary (206 words of 40-bits each). Estimated performance data, for this implementation of the fixed record length text compression hardware, are shown in **Fig. 12.**

The byte-organised variable record length APP could be implemented with the MOS CAM design described in Lea (1977a). This device known as the Micro-APP, incorporates the CAM design described in Lea (1975a) and would provide 32 words of 12 bits each in a 40-pin plastic dual-in-line pack. Hence 22 of these packs would be required to implement the *n*-gram dictionary (700 words of 12-bits each). Estimated performance data, for this implementation of the variable record length text compression hardware, are shown in **Fig. 13.**

A conventional implementation of the text compression hardware was also considered. This system comprised the DEC LSI 11 microprocessor with 4K words of both Random Access Memory (RAM) and Read Only Memory (ROM) support. Comparisons of the estimated costs of materials and the operating speeds of the three text compression hardware

Table 7 Average instruction execution counts and storage requirements of the three text compression systems

Average instruction execution count per character

	Conventional processor	Fixed record length APP	Variable record length APP
Compression	45	2·5	4·8
Decompression	12	1·8	4·3

Dictionary storage requirements (K-bits)

	Conventional processor	Fixed record length APP	Variable record length APP
Compression	15	8·2	9·0
Decompression*	12	—	—

*The APP uses the same dictionary for both compression and decompression tasks

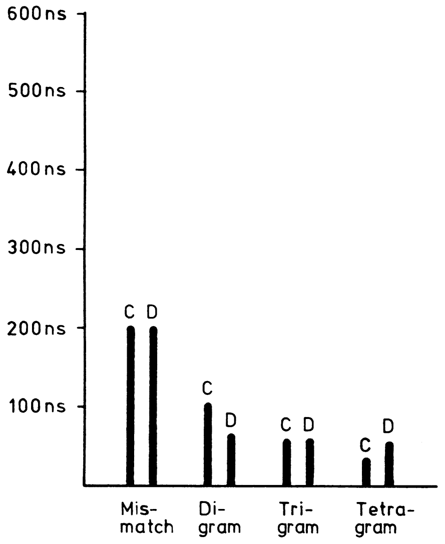


Fig. 12 Instruction execution times per character for text compression (C) and decompression (D) with the fixed record length APP hardware

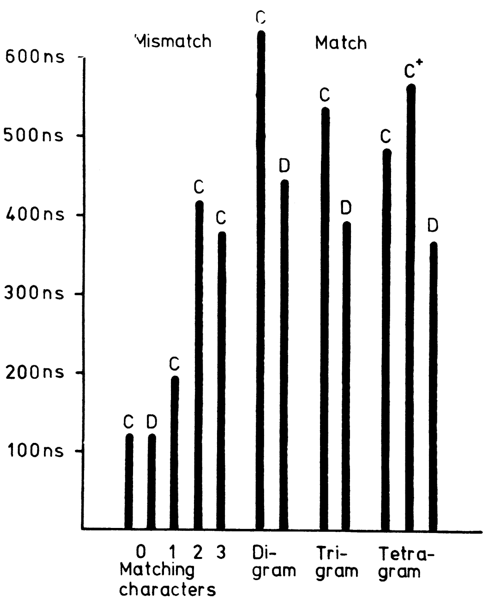


Fig. 13 Instruction execution times per character for text compression (C) and decompression (D) with the byte-organised variable record length APP hardware

Table 8 Estimated costs of materials for the three text compression hardware implementations

Estimated costs of materials

	Conventional processor	Fixed record length APP	Variable record length APP
AMA	(£211)[1]	£ 38*	£ 50*
WCL	(£124)[2]	£161	—
BCL (including DIR & DOR)		£ 57	£ 16
MPS (including TR)	(£ 59)[3]	£ 80†	£ 92†
IP, OP, CHQ, SW1/2	£ 36	£ 36	£ 36
Assembly (pcbs‡ & connectors)	£ 87	£116	£ 28
Overall System	£517	£488	£222

Assuming 300+ quantities

[1]4K RAM store [2]DEC LSI-11 [3]4K ROM store

*Assuming plastic dual-in-line packaging and *no* contribution towards LSI chip development costs

†Assuming maximum speed of operation

‡pcb = printed-circuit-board (with plated-through-holes)

Table 9 Average execution times for compression and decompression for the three text compression implementations

Average execution times per character

	Conventional processor	Fixed record length APP	Variable record length APP
Compression	360 μs	93 ns	594 ns
Decompression	96 μs	61 ns	418 ns

Table 10 Summary of character transmission rates and costs for the three text compression implementations

Character transmission rates (bytes-per-sec)

	Conventional processor	Fixed record length APP	Variable record length APP
Compression	1·1 K	4·1 M	0·64 M
Decompression	4·0 K	6·3 M	0·91 M
Processor Cost	£520	£490	£220

implementations are shown in **Tables 8** and **9** respectively. The figures in Table 9 are derived from the results of Table 7 and estimates of the instruction execution time for each implementation.

5.1. *Additional comments*

The cost estimates for the two APP implementations of text compression hardware assume the maximum speed of operation for each APP. Where lower levels of performance can be allowed the cost of the Micro-Program Store (MPS) can be reduced considerably, for example a 50% reduction in operating speed would allow a 50% reduction in the cost of the MPS unit.

It can be observed from Tables 8 and 9 that the variable record length text compression hardware is significantly cheaper and slower than the fixed record length hardware. This is due to the incorporation of the Word Control Logic (WCL) and the Associative Memory Array (AMA) within a single Micro-APP chip (Lea, 1977a). This device is based on the MOS CAM design described in Lea (1975a). A cheaper and faster version of the Micro-APP could be produced if its design were based on the MOS CAM design described in Lea (1975b) instead.

The Micro-APP design is not suitable for the implementation of the fixed record length APP hardware described in this paper. However the same LSI chip design technique could be applied to fixed record length text compression hardware in which the *n*-gram dictionary is restricted to:

1. A digram system, as in the systems described in Snyderman and Hunt (1970); Schieber and Thomas (1971) and Jewell (1976).
2. A trigram system with a permanent (read-only) *n*-gram dictionary.

Such systems could be implemented at a similar cost to that of the variable record length text compression hardware but would operate at a slightly lower speed than the fixed record length hardware.

6. Conclusions

Text compression using a coding dictionary of 200+ *n*-grams can provide a reduction of nearly 50% in file storage costs and an increase of nearly 100% in channel transmission speeds. However these advantages are offset by the storage requirement and execution time of the compression and decompression routines.

The application of special purpose hardware, incorporating an Associative Parallel Processor (APP), to text compression and decompression tasks has been reported in this paper. Two different implementations, based on the fixed record length and the byte-organised variable record length modes of data organisation, have been compared with a conventional microprocessor implementation for text compression hardware. The results of this work are summarised in Tables 7, 8 and 9. These results are not optimum but they are indicative of the considerable processing advantages of APP hardware over conventional microprocessor hardware for text compression and decompression tasks.

In contrast to the microprocessor alternative both APP systems can support realistic data transmission rates. With reference to Figs. 12 and 13; the worst case compression and decompression rates for the fixed record length hardware are 5M bytes-per-sec. and for the variable record length hardware are 1·6M bytes-per-sec. and 2·2M bytes-per-sec. respectively. A more meaningful assessment of performance is in terms of the maximum data rate of the disk which the text compressor/decompressor module can sustain without loss of synchronism. It is essential that the data flow, to and from a block of contiguous *n*-gram codes, is uninterrupted, to avoid losses in execution speed and/or the achieved degree of compression. Hence, the performance of the three implementations is summarised in these terms in **Table 10**. These results show that the APP implementations can accommodate the data rates of many disk systems.

Although the speed of the variable record length hardware can be improved (see Section 5.1) the fixed record length hardware has a definite speed advantage. By restricting the latter to digrams (or read-only digrams and trigrams) its cost can be reduced to compare more favourably with the former. However for higher order *n*-grams the variable record length hardware becomes progressively more attractive.

It can be seen from Table 8 that the cost of the Associative Memory Array AMA, is not the major component of the cost of the variable record-length hardware. Thus longer *n*-grams can be incorporated in the coding dictionary with much less than a pro-rata increase in cost. This may be attractive for increasing the degree of compression by including the most

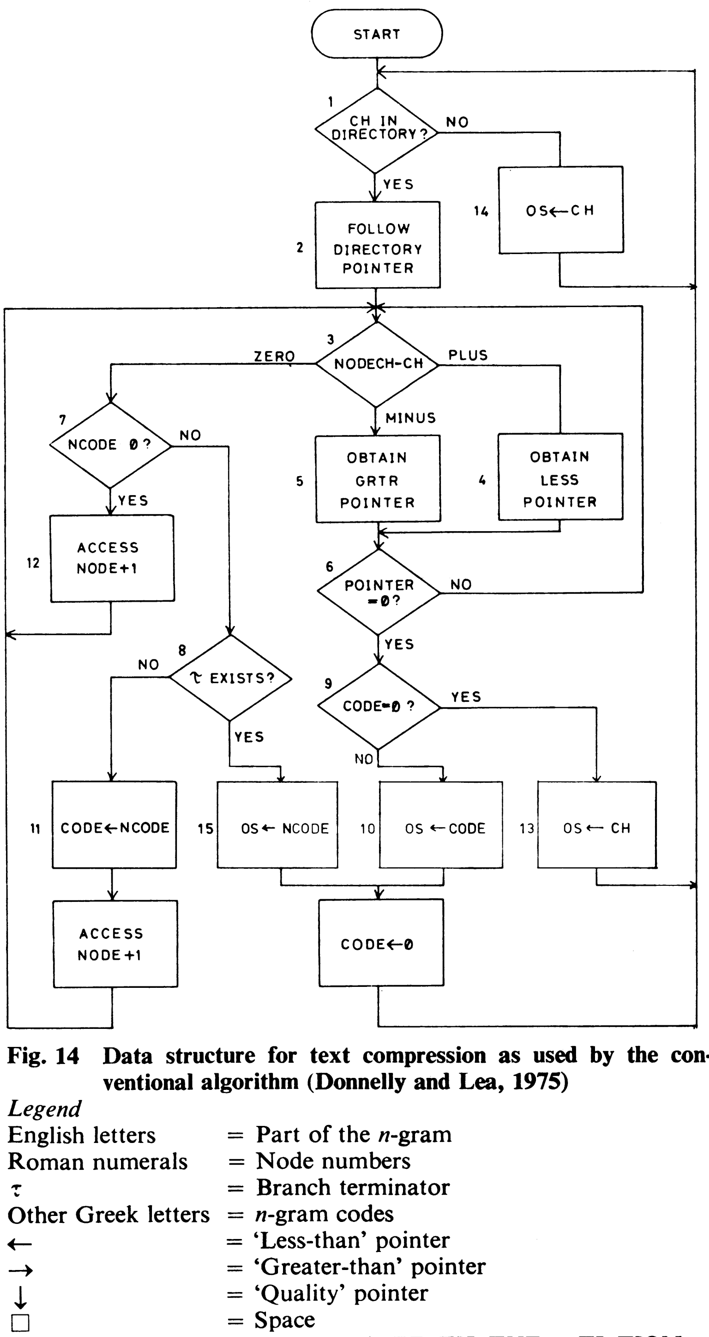


Fig. 14 Data structure for text compression as used by the conventional algorithm (Donnelly and Lea, 1975)

Legend

English letters	= Part of the *n*-gram
Roman numerals	= Node numbers
τ	= Branch terminator
Other Greek letters	= *n*-gram codes
←	= 'Less-than' pointer
→	= 'Greater-than' pointer
↓	= 'Quality' pointer
□	= Space
dictionary entry	= *n*-grams TA, TE, TH, THE□, TI, TION, TO, TR, TS, TU

frequent words within the longer *n*-grams. Hence if the speed of the variable record-length implementation is acceptable then it can be regarded as being better suited to text compression and decompression tasks.

The investigation has shown that the APP is well suited to text fragment encoding and decoding. Moreover the results of Table 10 indicate that a text compressor/decompressor, incorporating an APP, would effectively double the capacity and performance of a file-store for only a small extra cost. However the investigation has assumed:

1. the availability of the appropriate associative memory building block. Research leading to the development of cost-effective LSI CAM (Lea, 1975a; 1975b; 1976; 1977b) and Micro-APP (Lea, 1977a) chips is being undertaken at Brunel University.
2. adequate control of data transfer between the cpu and disk via the text compressor/decompressor module. Various control strategies have been considered. Ideally the module would be incorporated in the DMA channel between the cpu

and the disc controller (Donnelly, 1976b). However to avoid difficulties with maintenance contracts and software compatibility the implementation would be best left to the computer systems manufacturer. Alternatively the module could be interfaced to existing systems with a separate DMA channel. Obviously, the control strategy has a major impact on the cost-effectiveness of text compressor/decompressor modules.

Text compression systems, similar to those described in this paper, have been developed at Brunel University and evaluation studies are being carried out using INSPEC files for a realistic data base.

7. Acknowledgements
The author gratefully acknowledges the assistance of R. K. Donnelly in the preparation of this paper. Acknowledgement is also due to Professor M. F. Lynch (Sheffield University) who supplied the *n*-gram dictionary and to A. Negus (INSPEC) for the sample data-base.

The experimental associative processing facility at Brunel University was developed with the support of the Science Research Council. The investigation into the application of associative processing hardware to online text compression and decompression tasks is supported by the British Library. Patent protection for the two MOS Content Addressable Memory (CAM) designs (Lea, 1975a; 1975b; 1977b) and the Micro-APP design (Lea, 1977a) has been secured by the National Research Development Corporation (NRDC). An investigation into the feasibility of fabricating a Micro-APP using the Plessey Schottky I^2L process is being supported by the Advanced Computer Techniques Project (ACTP).

Appendix: Conventional Text Compression Algorithm

The text compression algorithm used for the conventional microprocessor implementation of the text compressor/decompressor module is similar to that reported by Byrne and Mullarney (1972) and Donnelly and Lea (1975).

The data structure on which the algorithm operates is shown in **Fig. 14** together with the set of *n*-grams which begin with the letter *T*. All characters which start *n*-grams have an entry in the directory, which is a pointer to a tree where the remainder of the *n*-grams are stored. Each node in the tree is allocated space for an *n*-gram character, a code, a branch-terminator, a 'greater than' pointer and a 'less than' pointer.

The algorithm which operates on this data structure is given in **Fig. 15**. It is assumed that text from the natural text domain is buffered in core and that there are two pointers associated with it:

1. a 'character-pointer' points to the current character being used to search the dictionary.
2. an '*n*-gram pointer' points to the start of the substring which is currently being processed.

With reference to Fig. 15, the mnemonic CH is used to denote a character but in order to simplify the presentation of the algorithm, the following conventions are used:

1. when CH is used during the dictionary search, the 'character-pointer' is used

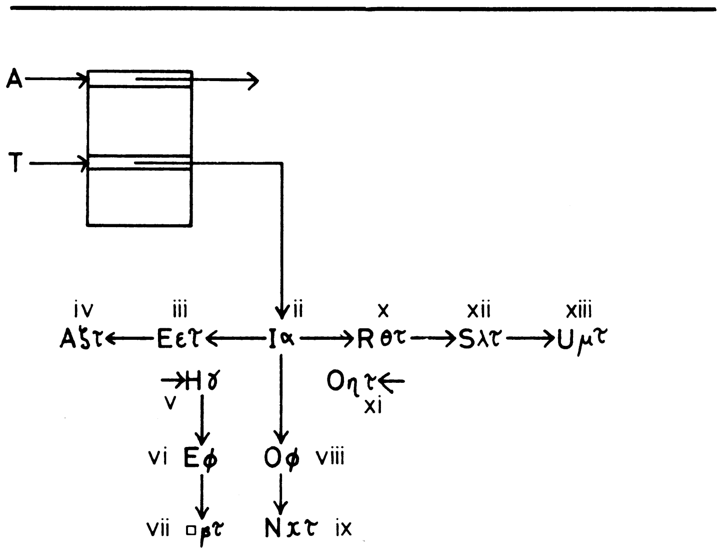


Fig. 15 A conventional text compression algorithm (Donnelly and Lea, 1975)

Legend

CH	= Current character
NODECH	= Character in current node
GRTR	= 'Greater-than' pointer
LESS	= 'Less-than' pointer
CODE	= Temporary storage
NCODE	= *n*-gram code
NODE + 1	= Implicit 'equality' pointer
OS	= Output string

Table 11 Operational sequence of the algorithm of Fig. 15 and data structure of Fig. 14.

CH	*nodes accessed and operations*
T	i (1, 2)
E	ii (3, 4, 6) iii (3, 7, 8, 10)
T	i (1, 2)
H	ii (3, 4, 6) iii (3, 5, 6) v (3, 7, 8, 11)
E	vi (3, 7, 12)
□	vii (3, 7, 8, 15)
T	i (1, 2)
W	ii (3, 5, 6) x (3, 5, 6) xii (3, 5, 6) xiii (3, 5, 6, 9, 13)

2. when the search for an *n*-gram is unsuccessful and a character is passed to the output string, the '*n*-gram pointer' is used.

The operational sequence of the algorithm is shown in **Table 11.**

As can be seen in Table 11, a considerable amount of time is taken up by processes other than successful character comparisons: even when an *n*-gram is found, there are likely to have been several unsuccessful comparisons and each time this happens a pointer must be obtained, tested and then followed. The use of location-address pointers gives rise to both storage and speed disadvantages. This problem can be eliminated when an Associative Parallel Processor is used for text compression.

References

Beaven, P. A. and Lewin, D. W. (1972). An associative parallel processing system for non-numerical computation, *The Computer Journal*, Vol. 15, pp. 343-349

Booth, A. D. (1967). A 'law' of occurrences for words of low frequency, *Inform. Control*, Vol. 70, pp. 386-393

Byrne, J. G. and Mullarney, A. (1972). A report on text compression prepared for INSPEC, Department of Computer Science, Trinity College, Dublin, Eire

Clare, A. C., Cook, E. M., and Lynch, M. F. (1972). The identification of variable length, equifrequent, character strings in a natural language data-base, *The Computer Journal*, Vol. 15, pp. 259-262

Colombo, D. S. and Rush, J. E. (1969). Use of word fragments in computer based retrieval systems, *J. Chem. Docum.*, Vol. 3, pp. 47-50

DODD, G. G. (1969). Elements of data management systems, *Computing Surveys*, Vol. 1, pp. 117-133

DONNELLY, R. K. and LEA, R. M. (1975). The application of an associative parallel processor to data compression for conventional file storage systems, Brunel University, Tech. Memo No. C/R/024

DONNELLY, R. K. (1976a). The design of a stand-alone associative processor for text compression, Brunel University Tech. Memo No. C/R/035

DONNELLY, R. K. (1976b). A study of a disc-based unit for text compression, Brunel University Tech. Memo No. C/R/036

DONNELLY, R. K. and MOTTERSHEAD, C. A. (1976). The internal operation of a compression and decompression unit which uses an associative memory, Brunel University, Tech. Memo. No. C/R/041

FAIRTHORNE, R. A. (1969). Empirical hyperbolic distribution (Bradford-Zipf-Mandelbrot) for bibliometric description and prediction, *J. Docum*, Vol. 25, pp. 319-343

FANO, R. M. (1961). *Transmission of information*, MIT Press and Wiley, Cambridge and New York

FOKKER, D. W. and LYNCH, M. F. (1974). Application of the variety-generator approach to searches of personal names in bibliographic data-bases—Part 1. Microstructure of personal authors names, *J. Lib. Autom.* Vol. 7, pp. 105-118

GILBERT, E. N. and MOORE, E. F. (1959). Variable length binary encoding, *Bell. Sys., Tech. J.*, Vol. 38, pp. 913-967

HANLON, A. G. (1966). Content addressable and associative memory systems—a survey, *IEEE trans.*, Vol. EC-15, pp. 509-521

HEAPS, H. S. (1972). Storage analysis of a compression coding for document data bases, *Information*, Vol. 10, pp. 47-61

HEAPS, H. S. and THIEL, L. H. (1970). Optimum procedures for economic information retrieval, *Inform. Stor. Retr.*, Vol. 6, pp. 137-153

HUFFMAN, D. A. (1952). A method for the construction of minimum redundancy codes, *Proc. IRE*, Vol. 40, pp. 1098-1101

JEWELL, G. C. (1976). Text compaction for information retrieval systems, *IEEE Systems, Man and Cybernetics Society Newsletter*, Vol. 5, pp. 4-7

LEA, R. M. and WRIGHT, J. S. (1973). A novel memory concept for information processing, *Datafair Research Papers*, Vol. II, pp. 413-417

LEA, R. M. (1975a). A design for a low-cost high-speed MOS associative memory, *The Radio and Electronic Engineer*, Vol. 45, No. 4, pp. 177-182

LEA, R. M. (1975b). Low-cost high-speed associative memory, *IEEE Journal of Solid-State Circuits*, SC-10, Vol. 3, pp. 179-181

LEA, R. M. (1975c). Information Processing with an Associative Parallel Processor, *IEEE Computer*, Vol. 8, pp. 25-32

LEA, R. M. (1976). The comparative cost of associative memory, *The Radio and Electronic Engineer*, Vol. 46, No. 10, pp. 487-497

LEA, R. M. (1977a). Micro-APP: A building block for low-cost high-speed Associative Parallel Processors, *The Radio and Electronic Engineer*, Vol. 47, No. 3, pp. 91-99

LEA, R. M. (1977b). Building blocks for associative memory, *Electronic Engineering*, Vol. 49, pp. 77-80

LEE, C. Y. (1962). Intercommunicating cells—a basis for a distributed logic computer, *Proc. AFIPS (FJCC)*, Vol. 22, pp. 130-136

LEFKOVITZ, D. (1969). *File structures for on-line systems*, Spartan Books

LYNCH, M. F. (1975). Variety generation—a reinterpretation of Shannon's mathematical theory of communication and its implications for Information Science, University of Sheffield

LYNCH, M. F., PETRIE, J. H. and SNELL, M. J. (1973). Analysis of the microstructure of titles in the INSPEC data-base, *Inform. Stor. Retr.*, Vol. 9, pp. 331-337

MINKER, J. (1971). An overview of associative or content-addressable memory systems and a KWIC index to the literature—1956-1970, *Computing Reviews*, Vol. 12, pp. 453-504

NOTLEY, M. G. (1970). The cumulative recurrence library, *The Computer Journal*, Vol. 13, pp. 14-19

PARHAMI, B. (1973). Associative memories and processors: an overview and selected bibliography, *Proc. IEEE*, Vol. 61, pp. 722-730

RUTH, S. S. and KREUTZER, P. J. (1972). Data compression for large business files, *Datamation*, pp. 62-66

SCHIEBER, W. D. and THOMAS, G. W. (1971). An algorithm for compaction of alphanumeric data, *J. of Library Automation*, Vol. 4, pp. 198-207

SCHUEGRAF, E. J. and HEAPS, H. S. (1973). Selection of equifrequent word fragments for information retrieval, *Inform. Stor. Retr.*, Vol. 9, pp. 697-711

SCHUEGRAF, E. J. and HEAPS, H. S. (1974). A comparison of algorithms for data-base compression by use of fragments as language elements, *Inform. Stor. Retr.*, Vol. 10, pp. 309-319

SCHWARTZ, E. S. and KLEIBOEMER, A. J. (1967). A language element for compression coding, *Inform. Control.*, Vol. 10, pp. 315-333

SHANNON, C. E. (1948). A mathematical theory of communication, *Bell Systems Tech. J.*, Vol. 27, pp. 379-423, pp. 623-656

SHANNON, C. E. (1951). Prediction and entropy of printed English, *Bell Sys. Tech. J.*, Vol. 30, pp. 50-64

SNYDERMAN, M. and HUNT, B. (1970). The myriad virtues of text compaction, *Datamation*, pp. 36-40

THIEL, L. H. and HEAPS, H. S. (1972). Program design for retrospective searches on large data bases, *Inform. Stor. Retr.*, Vol. 8, pp. 1-20

WAGNER, R. A. (1973). Common phrases and minimum space text storage, *CACM*, Vol. 16, pp. 148-153

WALKER, V. R. (1969). Compaction of names by *x*-grams, *Proc. Am. Soc. Inform. Sci.*, Vol. 6, pp. 129-135

WELLS, M. (1972). File compression using variable length encodings, *The Computer Journal*, Vol. 15, pp. 308-313

WHITE, H. E. (1967). Printed English compression by dictionary encoding, *Proc. IEEE*, Vol. 55, pp. 390-396

WILLIAMS, P. W. (1975). Criteria for choosing subsets to obtain maximum relative entropy, Dept. of Comput., UMIST