AN ALGORITHM FOR COMPACTION OF ALPHANUMERIC DATA

William D. SCHIEBER, George W. THOMAS: Central Library and Documentation Branch, International Labour Office, Geneva, Switzerland

Description of a technique for compressing data to be placed in computer auxiliary storage. The technique operates on the principle of taking two alphabetic characters frequently used in combination and replacing them with one unused special character code. Such one-for-two replacement has enabled the ILO to achieve a rate of compression of 43.5% on a data base of approximately 40,000 bibliographic records.

INTRODUCTION

This paper describes a technique for compacting alphanumeric data of the type found in bibliographic records. The file used for experimentation is that of the Central Library and Documentation Branch of the International Labour Office, Geneva, where approximately 40,000 bibliographic records are maintained on line for searches done by the Library for its clients. Work on the project was initiated in response to economic pressure to conserve direct-access storage space taken by this particularly large file. In studying the problem of how to effect compaction, several alternatives were considered.

The first was a recursive bit-pattern recognition technique of the type developed by DeMaine (1,2), which operates independently of the data to be compressed. This approach was rejected because of the apparent complexity of the coding and decoding algorithms, and also because early analyses indicated that further development of the second type of approach might ultimately yield higher compression ratios.

The second type of approach involves the replacement, by shorter non-data strings, of longer character strings known to exist with a high frequency in the data. This technique is data dependent and requires an analysis of what is to be encoded.

One such method is to separate words into their component parts: prefixes, stems and suffixes; and to effect compression by replacing these components with shorter codes. There have been several successful algorithms for separating words into their components. Salton (3) has done this in connection with his work on automatic indexing. Resnikoff and Dolby (4,5) have also examined the problem of word analysis in English for computational linguistics. Although this method appears to be viable as the basis of a compaction scheme, it was here excluded because ILO data was in several languages. Moreover, Dolby and Resnikoff's encoding and decoding routines require programs that perform extensive word analysis and dictionary look-up procedures that ILO was not in a position to develop.

The actual requirements observed were twofold: that the analysis of what strings were to be encoded be kept relatively simple, and that the encoding algorithm must combine simplicity and speed presumably by minimizing the amount of dictionary look-up required to encode and decode the selected string.

One of the most straightforward examples of the use of this technique is the work done by Snyderman and Hunt (6) that involves replacement of two data characters by single unused computer codes. However, the algorithm used by them does not base the selection of these two-character pairs (called "digrams") on their frequency of occurrence in the data. The technique described here is an attempt to improve and extend the concept by encoding digrams on the basis of frequency. The possibility of encoding longer character strings is also examined.

Three other related discussions of data compaction appear in papers by Myers et al. (7) and by DeMaine and his colleagues (8,9).

THE COMPRESSION TECHNIQUE

The basic technique used to compact the data file specifies that the most-frequently occurring digrams be replaced by single unused special-character codes. On an eight-bit character machine of the type used, there are a total of 256 possible character codes (bytes). Of this total only a small number are allocated to graphics (that is, characters which can be reproduced by the computer's printer). In addition, not all of the graphics provided for by the computer manufacturer appear in the user's data base. Thus, of the total code set, a large portion may go unused. Characters that are unallocated may be used to represent longer character strings. The most elementary form of substitution is the replacement of specific digrams. If these digrams can be selected on the basis of frequency, the compression ratio will be better than if selection is done independent of frequency.

This requires a frequency count of all digrams appearing in the data, and a subsequent ranking in order of decreasing frequency. Once the base character set is defined, and the digrams eligible for replacement are selected, the algorithm can be applied to any string of text.

The algorithm consists of two elements: encoding and decoding.

In encoding, the string to be encoded is examined from left to right. The initial character is examined to determine if it is the first of any encodable digram. If it is not, it is moved unchanged to the output area. If it is a possible candidate, the following character is checked against a table to verify whether or not this character pair can be replaced. If replacement can be effected, the code representing the digram is moved to the output area. If not, the algorithm then moves on to treat the second character in precisely the same way as the first. The algorithm continues, character-by-character until the entire string has been encoded. Following is a step-by-step description of the element.

1) Load length of string into a counter.
2) Set pointer to first character in string.
3) Check to determine whether character pointed can occur in combination. If character does not occur in combination, point to next character and repeat step 3.
4) If character can occur in combination, check following character in a table of valid combinations with the first character. If the digram cannot be encoded, advance pointer to next character and return to step 3.
5) If the digram is codable, move preceeding non-codable characters (if any) to output area, followed by the internal storage code for the digram.
6) Decrease the string length counter by one, advance pointer *two* positions beyond current value and return to step 3.

In the following example assume that only three digrams are defined as codable: AB, BE and DE. Assume also that the clear text to be encoded is the six-character string ABCDEF. After encoding the coded string would appear as:

AB C DE F
— · — ·

A horizontal line is used to represent a coded pair, a dot shows a single (non-combined) character. The encoded string above is of length four. Note that although BC was defined as an encodable digram, it did not combine in the example above because the digram AB was already encoded as a pair. The characters C and F do not combine, so they remain uncoded.

Note also that if the digram AB had not been defined as codable, the resultant combination would have been different in this case:

A BC DE F
· — — ·

The decoding algorithm serves to expand a compressed string so that the record can be displayed or printed. As in the encoding routines, decoding of the string goes from left to right. Bytes in the source string are examined one by one. If the code represents a single character, the print code for that character is moved to the output string. If the code represents a digram, the digram is moved to the output string. Decoding proceeds byte-by-byte as follows until end of string is reached:

1) Load string length into counter.
2) Set pointer to first byte in record.
3) Test character.
   If the code represents a single character, point to next source byte and retest.
4) If the code represents a digram:
   move all bytes (if any) up to the coded digram; and move in the digram.
5) Increase the length value by one, point to next source byte and continue with step 3.

APPLICATION OF THE TECHNIQUE

The algorithm, when used on the data base of approximately 40,000 records was found to yield 43.5% compaction. The file contains bibliographic records of the type shown in Figure 1.

```
41345          .          1970                    70A1350
WARNER M
STONE M
THE DATA BANK SOCIETY - ORGANIZATIONS, COMPUTERS AND SOCIAL
FREEDOM.
LONDON, GEORGE ALLEN AND UNWIN, <1970>. 244 P. CHARTS.
   /SOCIAL RESEARCH/ INTO THE POTENTIAL THREAT TO PRIVACY
   AND FREEDOM (/HUMAN RIGHT/S) THROUGH THE MISUSE OF /DATA
   BANK/S - EXAMINES /COMPUTER/ BASED /INFORMATION
   RETRIEVAL/, THE IMPACT OF COMPUTER TECHNOLOGY ON
   BRANCHES OF THE /PUBLIC ADMINISTRATION/ AND /HEALTH
   SERVICE/S IN THE /USA/ AND THE /UK/ AND CONCLUDES THAT,
   IN ORDER TO PROTECT HUMAN DIGNITY, THE NEW POWERS MUST
   BE KEPT IN CHECK. /BIBLIOGRAPHY/ PP. 236 TO 242 AND
   /REFERENCE/S.
ENGL
```

Fig. 1. Sample Record from Test File.

Each record contains a bibliographic segment as well as a brief abstract containing descriptors placed between slashes for computer identification. A large amount of blank space appears on the printed version of these records; however, the uncoded machine readable copy does not contain blanks, except between words and as filler characters in the few fields defined as fixed-length. The average length of a record is 535 characters (10).

The valid graphics appearing in the data are shown in Table 1, along with the percentage of occurrence of each character throughout the entire file.

Table 1. Single-Character Frequency

Graphic	Freq. %	Graphic	Freq. %	Graphic	Freq. %	Graphic	Freq. %	Graphic	Freq. %
♭	14.87	/	4.32	H	1.58	–	0.63	8	0.31
E	7.63	C	3.48	.	1.52	W	0.50	(	0.28
N	6.38	L	3.32	,	1.52	2	0.42	)	0.28
I	6.01	D	2.32	1	1.08	K	0.42	+	0.21
A	6.01	U	2.21	V	0.91	3	0.40	J	0.15
∅	5.86	P	2.12	B	0.87	5	0.37	X	0.14
T	5.50	M	2.02	9	0.83	7	0.37	Z	0.13
R	4.82	F	1.61	Y	0.82	0	0.35	Q	0.08
S	4.61	G	1.58	6	0.81	4	0.34	Misc. Spec.	0.01

As might be expected, the blank (♭) occurs most frequently in the data because of its use as a word separator. The slash occurs more frequently than is normal because of its special use as a descriptor delimiter. It should also be noted that the data contains no lower-case characters. This is advantageous to the algorithm because it considerably lessens the total number of possible digram combinations. As a result, a larger proportion of the file is codable in the limited set chosen as codable pairs, and because the absence of 26 graphics allows the inclusion of 26 additional coded pairs.

In the file used for compaction there are 58 valid graphics. Allowing one character for special functions leaves 197 unallocated character codes (of a total of 256 possible). A digram frequency analysis was performed on the entire file and the digrams ranked in order of decreasing frequency. From this list the first 197 digrams were selected as those which were eligible for replacement by single-character codes. Table 2 shows these "encodable" digrams arranged by lead character.

The algorithm was programmed in Assembler language for use on an IBM 360/40 computer. The encoding element requires approximately 8,000 bytes of main storage; the decoding element requires approximately 2,000 bytes. In order to obtain data on the amount of computer time required to encode and decode the file, the following tests were performed. To find the encoding time, the file was loaded from tape to disk. The tape copy of the file was uncoded, the disk copy compacted. Loading time for 41,839 records was 52 minutes and 51 seconds. The same tape to disk operation without encoding took 28:08. The time difference (24:43) represents encoding time for 41,839 records, or .035 seconds per record.

A decoding test was done by unloading the previously coded disk file to tape. The time taken was 41:52, versus a time of 20:20 for unloading

an uncompacted file. The time difference (21:32) represents decoding time for 41,839 records, or .031 seconds per record.

The compaction ratio, as indicated above, was 43.5 per cent. For purposes of comparison, the algorithm developed by Snyderman and Hunt (6) was tested and found to yield a compaction ratio of 32.5% when applied to the same data file.

Table 2. Most Frequently Occuring Digrams

Lead Char.	*Eligible Digrams*
A	AB AC AD AG AI AL AM AN AP AR AS AT Ab
B	BL BO
C	CA CE CH CI CL CO CT CU Cb C.
D	DE DI DU Db D/
E	EA EC ED EF EL EM EN EP ER ES ET EV Eb E/
F	FE FI FO FR Fb
G	GE GL GR Gb G/
H	HA HE HI HO Hb
I	IA IC IE IL IN IO IS IT IV
L	LA LE LI LL LO LU Lb
M	MA ME MI MM MU Mb
N	NA NC ND NE NG NI NO NS NT Nb N/
O	OC OD OF OG OL OM ON OP OR OU OV Ob
P	PA PE PL PO PR P.
R	RA RE RI RK RN RO RS RT RU RY Rb R/
S	SA SE SI SO SP SS ST SU Sb S, S.
T	TA TC TE TH TI TO TR TS TU TY Tb T/
U	UC UD UL UN UR US UT
V	VA VE VI
W	WO
Y	Yb Y/
b	bA bB bC bD bE bG bI bL bM bN bO bP bR bS bT bU bW bb b/ b– b(
1	19
/	/A /C /E /I /L /M /P /R /S /T /b /,
,	,b
.	.b
–	–b
)	),

POSSIBLE EXTENSION OF THE ALGORITHM

Currently the compression technique encodes only pairs of characters. There might be good reason to extend the technique to the encoding of longer strings—provided a significantly higher compaction ratio could be

achieved without undue increase in processing time. One could consider encoding trigrams, quadrigrams, and up to *n*-grams. The English word "the", for example, may occur often enough in the data to make it worth coding.

The arguments against encoding longer strings are several. Prime among these is the difficulty of deciding what is to be encoded. Doing an analysis of digrams is a relatively straightforward affair, whereas an analysis of trigrams and longer strings is considerably more costly, because of the fact that there are more combinations. Furthermore, if longer strings are to be encoded, the algorithms for encoding and decoding become more complex and time-consuming to employ.

One approach to this type of extension is to take a particular type of character string, namely a word, and to encode certain words which appear frequently. A test of this technique was made to encode particular words in the data: descriptors. All descriptors (about 1200 in number) appear specially marked by slashes in the abstract field of the record. Each descriptor (including the slashes) was replaced by a two-character code. After replacement, the normal compaction algorithm was applied to the record. A compaction ratio of 56.4% was obtained when encoding a small sample of twenty records (10,777 characters).

The specific difficulty anticipated in this extension is the amount of either processing time or storage space which the decoding routines would require. If the look-up table for the actual descriptor values were to be located on disk, the time to retrieve and decode each record might be rather long. On the other hand, if the look-up table were to be in main storage at the time of processing, its size might exclude the ability to do anything else, particularly when on-line retrieval is done in an extremely limited amount of main storage area. A partial solution to this problem might be to keep the look-up tables for the most frequently occurring terms in main storage and the others on disk. At present further analysis is being done to determine the value of this approach.

CONCLUSIONS

The compaction algorithm performs relatively efficiently given the type of data used in text data base (i.e. data without lower case alphabetics, having a limited number of special characters, in primarily English text). The times for decoding individual records (.031 sec/record) indicate that on a normal print or terminal display operation, no noticeable increase in access time will be incurred. However several types of problems are encountered when treating other kinds of data.

Since the algorithm works on the basis of replacing the most-frequently occurring *n*-grams by single-byte codes, the compaction ratio is dependent on the number of codes that can be "freed up" for *n*-gram representation. The more codes that can be reallocated to *n*-grams, the better the compaction. Data which would pose complications to the algorithm—as currently defined—can be separated for discussion as follows:

1) data containing both upper *and* lower case characters (as well as a limited set of special characters), and 2) data which might possibly contain a wide variety of little-used special graphics.

If lower-case characters are used, a possible way to encode data using this technique is to harken back to the time-honored method of representing lower-case with upper-case codes, and upper-case characters by their value, preceeded by a single shift code (e.g., #ACCESS for Access). The shift code blank character digram would undoubtedly figure relatively high on the frequency list, making it eligible as an encodable digram.

The second problem occurs when one attempts to compact data having a large set of graphics. A good example of this is bibliographic data containing a wide variety of little-used characters of the type now being provided for in the MARC tapes (11) issued by the U. S. Library of Congress (such as the Icelandic Thorn). Normally representation of these graphics is done by allocating as many codes as required from the possible 256-code set. Since the compaction ratio is dependent on the number of unallocated internal codes, a possible solution to this dilemma might be to represent little-used graphics by multi-byte codes which would free the codes for representation of frequently occurring *n*-grams.

Further, it is noticeable that the more homogeneous the data the higher the compression ratio. This means that data all in one language will encode better than data in many languages. There is, unfortunately, no ready solution to this problem, given the constraints of this algorithm. In dealing with heterogeneous data one must be prepared to accept a lower compression factor.

Without doubt to be able to effect a savings of around 40% for storage space is significant. The price for this ability is computer processing time, and the more complex the encoding and decoding routines, the more time is required. There is a calculable break-even point at which it becomes economically more attractive to buy x amount of additional storage space than to spend the equivalent cost on data compaction. Yet at the present cost of direct-access storage, compaction may be a possible solution for organizations with large data files.

REFERENCES

1. Marron, B. A.; DeMaine, P. A. D.: "Automatic Data Compression," *Communications of the ACM*, 10 (November 1967), 711-715.
2. DeMaine, P. A. D.; Kloss, K.; Marron, B. A.: *The SOLID System III: Alphanumeric Compression.* (Washington, D. C.: National Bureau of Standards, 1967). (Technical Note 413).
3. Salton, G.: *Automatic Information Organization and Retrieval* (New York: McGraw-Hill, 1968).
4. Resnikoff, H. L.; Dolby, J. L.: "The Nature of Affixing in Written English," *Mechanical Translation*, 8 (March 1965), 84-89.

5. Resnikoff, H. L.; Dolby, J. L.: "The Nature of Affixing in Written English," *Mechanical Translation,* 9 (June 1966), 23-33.
6. Snyderman, Martin; Hunt, Bernard: "The Myriad Virtues of Text Compaction," *Datamation* (December 1, 1970), 36-40.
7. Myers, W.; Townsend, M.; Townsend, T.: "Data Compression by Hardware or Software," *Datamation* (April 1966), 39-43.
8. DeMaine, P. A. D.; Kloss, K.; Marron, B. A.: *The SOLID System II. Numeric Compression.* (Washington, D. C.: National Bureau of Standards, 1967). (Technical Note 413).
9. DeMaine, P. A. D.; Marron, B. A.: "The SOLID System I. A Method for Organizing and Searching Files." In Schecter, G. (Ed.): *Information Retrieval—A Critical View.* (Washington, D. C.: Thompson Book Co., 1967).
10. Schieber, W.: *ISIS (Integrated Scientific Information System; A General Description of an Approach to Computerized Bibliographical Control).* (Geneva: International Labour Office, 1971).
11. *Books: A MARC Format; Specification of Magnetic Tapes Containing Monographic Catalog Records in the MARC II Format.* (Washington, D. C.: Library of Congress, Information Systems Office, 1970.)