Sci. Annals, Fac. Phys. & Mathem., Univ. Thessaloniki, **18**, 73 (1978)

MINIMISING STORAGE SPACE REQUIREMENTS IN CRITICAL CASES

by

C. LAZOS and A. VAFIADIS

(Department of Mathematics, Aristotelian University of Thessaloniki)

(Received 30.6.78)

Abstract: *Two methods of compacting textual data, to reduce computer storage requirements. are described. It is shown that where a high degree of homogeneity exists in such text, substantial storage savings can be realised. A case study of a data-base of Greek surnames and first names is used as an example.*

1. INTRODUCTION

A significant advantage of any computer system is its ability to store and process vast volumes of data. While it is readily recognised that any costing includes processing costs, the cost of actually storing data in computer readable form is not insignificant, but often overlooked. It is the storage space required to store large quantities of data, that are inevitably associated with computer based systems, that make such costs important and puts a high premium on minimising the total storage space required, particularly in on-line operations where there is limited physical storage available. Two types of on-line storage devices are present in almost all computer systems; the main memory storage, so called because of its role of serving the central processing unit itself, with the characteristic of high speed of storing or retrieving data, but with its associated high cost per unit of storage and consequent limitation in physical capacity—typically thousands to hundreds of thousands of characters (one character of storage is often referred to as one byte). In contrast, the second form of storage, secondary storage, has capacity several orders of magnitude larger and with costs considerably less per unit of storage. Magnetic discs and magnetic drums are examples of suuch storage media. Despite such capacity, tens of millions of characters, typical applications often face the problem of saturating the available

space or are handicapped by a limitation to the amount of space the installation is able to allocate for it. Any factors reducing storage space requirements are therefore important.

Minimising the storage space requirements for a given data-base has been studied at various levels. For instance, Wagner, in order to reduce storage used to handle error messages for the PL/C compiler, has hand selected the most common error messages and compacted them using a specially written algorithm[1,2]. Alternatively, Snyderman[3], very intelligently took advantage of the unused sections of the hexadecimal EBCDIC codes for characters, to store in one character many of the most frequently used pairs of characters found in text data.

2. STORAGE SAVING

Digital computers ultimately store data in some coded form i.e. data is coded and stored as a sequence of symbols (ultimately a string of binary bits — logically 0's and 1's). However, it is often more convenient to consider characters coded in another form. For instance, octal (base 8) orientated machines permit a combination of up to $n = 64$ different characters (2 to the power of 6) while hexadecimal (base 16) orientated computers allow $n = 256$ character set (2 power of 8). In both these cases and other representations, it is often possible, depending on the data, to save storage by encoding the characters in a more compact way i.e. retain uniqueness of identification while reducing the length of storing of binary bits required for representation. Such a technique is often possible where the data consists of a subset of the normal character set available thus permitting a reduction in the number of bits of information necessary to store each of them.

If S is the universal set of all characters permitted in a given computer system and S_1, the subset of characters used in a given data-base then

$$S_1 \subset S$$

If N is the number of characters in S, then to store any character, say c_i ($c_i \in S$) the number of bits n required, must satisfy the relation:

$$n \geq \lfloor \log_2(N-1) \rfloor + 1$$

with $n = \lfloor \log_2(N-1 \rfloor + 1$ being the minimum.

Thus if N is reduced from 64 to the range $16 < N_1 \leq 32$ then the minimum number of bits is given by

$$n = \lfloor \log_2(N_1 - 1) \rfloor + 1 = 4 + 1 = 5$$

This in turn implies that fot a 6 bit (octal) machine, a saving of 17% in storage could be achieved while for an 8 bit (hexadecimal) computer it could be as much as 37.5%. This is the lowest level one could encode the character subset and although quite feasible, a greater saving could be realised by encoding not at the character level but at a higher level i.e. groups of characters (words) or fields. Better still, a combination of both the techniques. However, it must not be ignored that a large data-base is organised in the form of records where several such records are grouped together to form a block. Each record in any block consists of one or more fields of data.

Now, if the text data contained in these records has a high degree of homogeneity in one or more of the record fields, space can be saved by numerically encoding the text in such fields. The encoding procedure would consist of storing the actual text from the appropriate field in one ordered list for that field and replacing the text by a corresponding numerical code, devised for each text e.g. the position of the text in the list. This numerical value, of course, occupies a certain amount of storage space. Suppose L represents the number of characters in the uncoded text for a given field and l the numerical code equivalent. Then the space saving is given by:

$$SV = 1 - l/L \quad ----- \quad (1)$$

where if $l < L$ then $SV < 0$
if $l = L$ then $SV = 0$
and if $l < L$ then $SV < 0$

This is the simplest and ideal case where all the text for the field can be encoded. In general, however, only part of the list of texts for a field can be encoded, the formula becomes:

$$SV = (1 - l/L)\, P \quad ----- \quad (2)$$

where P = ratio of texts encoded to total number of texts.

In this latter case, not all text of a field would be encoded leading to a situation where the data-base would contain variable length records.

3. ENCODING AND DECODING PROCEDURES

Let l be the length of the field (in characters) of the numerical code devised for that particular text. Then the maximum numerical code value that can be stored in l and therefore the maximum number of encoded texts is given by:

$$N = 2^{lm} - 1 \quad \text{----- (3)} \qquad (\text{where } ^{l} = l)$$

where m = number of bits needed to represent each character in the computer.

In the simple case, where all the different texts of a particular field can be encoded then the numerical code would be the position pointer of the text in the list of texts. Obviously the length of this list need not be N. In fact, it could well be less than N which would be the case in a considerable number of applications.

In the more complex case, where some of the texts would be encoded and some not, the procedure adopted could be to allocate sufficient space in the field to not only store the numerical code where appropriate but also to indicate:

(i) the text has not been encoded i.e. is not in the list.

(ii) the size of the text, in characters and that the text is stored in the record itself.

Thus assuming that the maximum size of a text is L characters, then formula (3) becomes:

$$N = 2^{lm} - 1 - L \quad \text{----- (4)}$$

In other words, the maximum size of the list of texts would be smaller by L, a fact that does not affect matters as will be shown in Section 5. In such a case, then, the following algorithm can be used for de-coding purposes:

IF numerical code $\leq 2^{lm} - 1 - L$

THEN [the text has been encorded and the numerical code represents the position of the full text in the text list]

ELSE DO [M = numerical code $- (2^{lm} - 1 - L)$ and the full text is the next M characters in the record]

3.1 Timing

It is clear that savings in space can be realised using character subsets, text field encoding or combinations of these methods. However, such savings are achieved at the cost of additional processing time necessary to encode and decode.

The time consuming aspect of the encoding process in text field coding is concerned with the reading of all records in the data-base, matching the text from the particular field with those stored in the text list and if a match is found, then to encode the field and write the encoded (and smaller) record into the data-base. Various methods can be used to minimise the search of the text list e.g. linear search, binary search, text transformation addressing. The latter, more familiarly referred to as Hash coding[4], is probably the more appropriate method in the circumstances as it is related to the specific text data being handled, although it does have the disadvantage of requiring additional storage over and above that necessary to hold the actual text. However, the overhead time necessary to encode data-base records can be minimised by efficient searching and it need only be performed once for each record present in the data-base.

In contrast the de-coding time is of little consequence in that the method of encoding suggested permits the numerical code to be used to locate the full text of the field directly in the text list.

4. A CASE STUDY

In many commercial data processing applications it is quite common to observe data-base records constituted from surnames, first names, married names, etc., together with other information of a numerical as well as textual nature. In practice the design of such records involve allocating a fixed number of character storage positions for each text field in the record e.g. a field capable of storing 20 characters may be reserved for the surname of the person whose record is being constructed. While this fixed capacity of fields in records leads to a simplicity of record structure (all records being of the same structure and size) and processing, it can lead to two drawbacks.

(a) If, say, a surname of any textual data has more characters than the field in which it is to be stored has been allocated, then the surname etc., is usually truncated by the necessary number of

characters to permit its storage, in the designated field.

(b) A solution to (a) is often to increase the number of characters reserved for a field in order to reduce the frequency of trunation, but this in turn aggravates the space wasted in a field occupied by, say a surname containing less characters than the capacity of its associated record field.

One solution to both these drawbacks is permitting records to vary in size as textual data does but this leads to increased complexity in processing records where each field could be of variable length and may even be empty. This solution is usually related to the special characteristics of the data-base e.g. several data items being absent or having a wide range in character length in a substantial number of the records.

In constrast to such data, there are many applications where there exists very little variation in the text data stored in each field and often a substantial portion of the text in a given field is exactly the same. For instance, many Greeks share the same surname or Christian name. Such similarity can be used to minimise storage requirements.

In order to discover the degree of similarity within Greek surnames and other names, the Ministry of Education was asked to provide a tape containing the names of all candidates registered for the 1976-77 examinations. The Ministry very kindly agreed and supplied a data-base file containing nearly 80,000 records of fixed length. Computer analysis of this large sample has produced the results show in Table 1.

TABLE 1.

SURNAMES		NAMES	
No of surnames	%	No of names	%
372	20	20	50
945	30	31	60
2001	40	48	70
3827	50	85	80
6435	60	182	90
9769	70	1893	100
35000	100		

For Greek surnames, a 16 to 20 character space field is usually

reserved in most applications and a 10 to 14 character field for first names. Using formula (1) given earlier, the space saving can be investigated as L is varied from 10 to 20 and l from 1 to 3. The results of this analysis is shown in Table 2.

TABLE 2.

L	$l = 1$	$l = 2$	$l = 3$
10	0.900	0.800	0.700
12	0.917	0.833	0.750
14	0.929	0.857	0.786
16	0.938	0.875	0.813
18	0.944	0.889	0.833
20	0.950	0.900	0.850

5. RESULTS

5.1 A consideration of first names

The data contained in Table 1 shows that 50% of the sample population share just 20 different names, while 1893 different names are found throughout the total population. The question therefore is how much space should be allocated to encode first names. An estimate of this can be obtained by using the results together with formulae (1) and (2) to derive the results shown in Table 3. This data has been calculated for an octal (64 different characters, each requiring 6 bits) orientated computer.

TABLE 3.

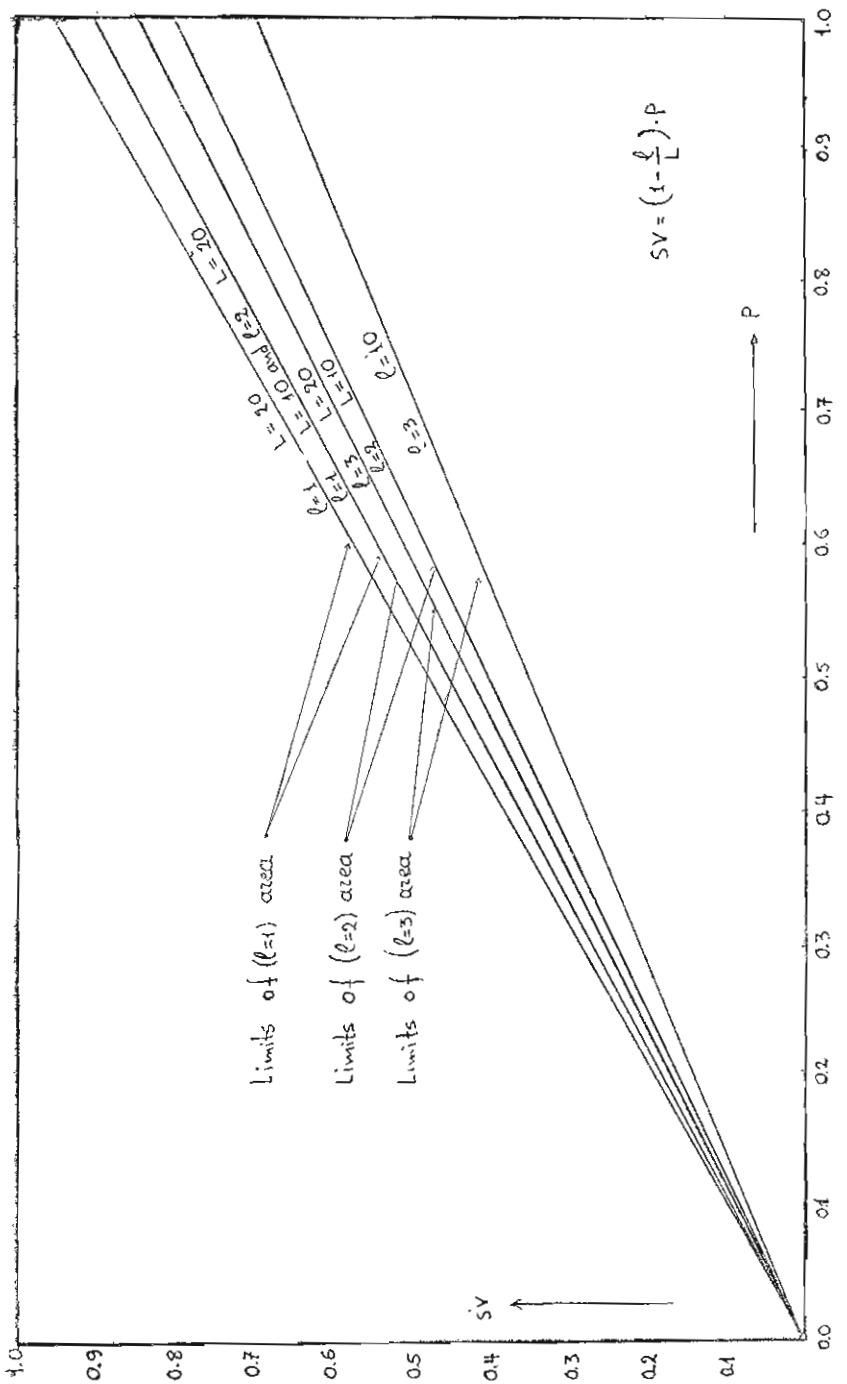
L	$l = 1$				$l = 2$			
	C	N	P	SV	C	N	P	SV
10	0.900	53	0.72	0.648	0.800	4085	1.00	0.800
12	0.917	51	0.71	0.651	0.833	4083	1.00	0.833
14	0.929	49	0.70	0.650	0.857	4081	1.00	0.857

where: $C = 1 - l/L$

N = Length of text list containing the actual names ($2^{lm} - 1 - L$)

P = Fraction of names stored in the text list

SV = Storage saving (maximum = 1.00)



An inspection of the data of Table 3 indicates that L = 12 (the length more frequently used for Greek first names), a text list of length 51 can cover up to 71% of names (P = 0.71) and realise a saving of up to 65.1%. Calculations for a hexadecimal machine (256 different characters, each requiring 8 bits) indicate that with N = 213, encoding could adequately cover 90% of the names with a consequent saving of up to 82.6%. Again the Table shows that for $l = 2$, encoding for octal machines can accommodate the entire spectrum of names and gave a total saving of up to 83.3%. Comparable results were also found for hexadecimal machines. These results are also shown in graphical form in Figure 1.

5.2 The Case of Surnames

An inspection of the data for surnames shown in Table 1 shows a considerably larger number of surnames shared by a smaller percentage of the population sample than found for first names. For example, 20% of the population share 372 surnames but only 20 first names. However, applying the same procedure to surnames as that applied to first names, data shown in Table 4 has been compiled. The results are also shown in graphical form in figure 1.

TABLE 4.

L	$l=1$				$l=2$				$l=3$			
	C	N	P	SV	C	N	P	SV	C	N	P	SV
10	.900	53	.068	.061	.800	4085	.51	.408	.700	262,133	1.0	.700
12	.917	51	.066	.061	.833	4083	.51	.425	.750	262,131	1.0	.750
14	.929	49	.065	.060	.855	4081	.51	.436	.786	262,129	1.0	.786
16	.938	47	.063	.059	.875	4079	.51	.446	.813	262,127	1.0	.813
18	.944	45	.062	.058	.889	4077	.51	.453	.833	262,125	1.0	.833
20	.950	43	.059	.056	.900	4075	.509	.458	.850	262,123	1.0	.850

Where: C, N, P and SV are the same as those defined in Table 3

These results indicate that for $l = 1$ i.e. 1 byte of storage, encoding of surnames would lead to little saving in space; specifically 5.6 to 6.1%. Alternatively, for $l = 3$, a considerable amount of storage is necessary for the text list of surnames (N = 262133). In practice a value of N = 35000 would be adequate but still unacceptable because actual storage

requirements would be 35000 X 20 = 700000 bytes of data. The case of $l = 2$ appears more promising. For an octal computer, the text list could accommodate more than 50% of the names and realise a saving of between 40.80 and 45.80%. A similar calculation for an hexadecimal computer shows that more than 50% could be stored; N in this case would be 35000 covering the whole spectrum of surnames. It has, of course, been pointed out earlier that this is still considered uneconomical. However, if a given computer system permits the allocation of core storage to allow N = 10000, this could be used to cover up to 70% of the range of surnames with a net saving in space of between 56 and 63%.

5.3 Overall Saving

In the more general aud more complex case where only parts of the text of a given field are encoded, the records will clearly be of variable length as described in Section 2; the encoded text parts would lead to a storage saving calculated from formulae (1) or (2). For the non-encoded text, a further saving can be obtained by analysis of the texts in order to reduce the fixed maximum space allocated for the field.

Thus if the encoding procedure requires l bytes for the numerical code, L is the maximum length reserved for the field in characters, while the average length of the actual text is l_1, then the overall saving in storage would be given by:

$$\mathrm{SAL} = 1 - \frac{l + (1-\mathrm{P})\, l_1}{\mathrm{L}}$$

Applying this formula to the case of the surnames where $l = 2$, $l_1 = 10$, L = 20 and P = 0.50 then; SAL = 0.65 i.e. 65% overall saving. Compare this with the saving of 45.80% for those surnames stored in the text list.

6. CONCLUSIONS

In conjunction with many other design criteria considered in data processing applications, the question of minimising storage for a database is a balance between saving of space and increased processing time; minimising storage requirements requires additional processing time to encode and decode the text along with increased complexity of program design and the associated development effort. In constrast, the

storage space saved can be considerable, leading to more records being stored in a given space and thereby leading to a reduction in the time to transfer records to/from the secondary storage. Since a characteristic of virtually all data processing systems is their computer execution time being composed almost entirely of time to transfer data, any saving in this aspect will be highly beneficial not only to the execution time for the system but also benefit the multiprogrammed systems in which such in which such jobs are invariably run.

In conclusion, it appears that in systems where space is critical then the procedures for saving storage outlined above would be beneficial despite the increase in complexity and processing time that results from the encoding and decoding procedures.

Acknowledgment

We express our gratitude to the Ministry of Education for kindly supplying the data-base of Greek names used in this project, and to Dr. J. R. Yandle, of the University of Birmingham, England for his useful comments.

REFERENCES

1. WAGNER, R. A., (March 1973): Communications of the ACM. «Common phrases and minimum space text storage».
2. WAGNER, R. A., (March 1973): Communications of the ACM. «An Algorithm for exatracting phrases in a space optimal fashion».
3. SNYDERMAN, M. and HANT, B. (December 1970): Datamation. «The virtues of text compaction».
4. HOPGOOD, F. R. A. (1968): «Compiling techniques», McDonald Computer Minographs.

ΠΕΡΙΛΗΨΗ

ΕΛΑΧΙΣΤΟΠΟΙΗΣΗ ΤΟΥ ΧΩΡΟΥ ΕΝΑΠΟΘΗΚΕΥΣΕΩΣ ΔΕΔΟΜΕΝΩΝ ΣΕ ΚΡΙΣΙΜΕΣ ΠΕΡΙΠΤΩΣΕΙΣ

Ὑπὸ

Κ. ΛΑΖΟΥ καὶ Α. ΒΑΦΕΙΑΔΗ

(Μαθηματικὸ τμῆμα, Ἀριστοτελείου Πανεπιστημίου Θεσσαλονίκης)

Ἐξετάζονται τρόποι μειώσεως τοῦ χώρου ἐναποθηκεύσεως δεδομένων σὲ δευτερεύουσες μνῆμες Ἠλεκτρονικῶν Ὑπολογιστῶν

Ἀποδεικνύεται ὅτι ἂν τὰ δεδομένα παρουσιάζουν μεγάλη ὁμοιομορφία σὲ ἕνα ἢ περισσότερους τομεῖς τότε μπορεῖ νὰ προκύψει σημαντικὴ μείωση τοῦ χώρου ἐναποθηκεύσεως. Ἂν καὶ γιὰ νὰ γίνει αὐτὸ χρειάζεται εἰδικὴ κωδικοποίηση καὶ ἀπαιτεῖται περισσότερος χρόνος τῆς κεντρικῆς μονάδος (CPU), παρ' ὅλα αὐτὰ τὰ ὀφέλη εἶναι σημαντικὰ γιατὶ ξέχωρα ἀπὸ τὴν ἐλάττωση τοῦ χώρου αὐτοῦ καθ' ἑαυτοῦ, θὰ μποροῦν νὰ βρίσκονται μέσα στὴν κεντρικὴ μνήμη περισσότερα δεδομένα, μὲ ἀποτέλεσμα τὴν ταχύτερη ἐπεξεργασία τους.

Συγκεκριμένη μελέτη μὲ ἑλληνικὰ ὀνόματα (κύρια καὶ μικρὰ) ἀπέδειξε ὅτι εἶναι δυνατὸ νὰ προκύψει σημαντικὸ ὄφελος χώρου.