COMPRESSION FOR AN EFFECTIVE MANAGEMENT OF TELEMETRY DATA[1]

J.-P. Arcangeli*
M. Crochemore**
J.-N. Hourcastagnou ***
J.-E. Pin****

*Institut de Recherche en Informatique de Toulouse & Greco de Programmation du C.N.R.S., Université P. Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex, France
**Laboratoire d'Informatique Théorique et Programmation & Greco de Programmation du C.N.R.S., Université Paris VII, 2 Place Jussieu, 75251 Paris Cedex 05, France
***Centre National d'Etudes Spatiales, TE/IS/PS/TD, 18 Avenue E. Belin, 31055 Toulouse Cedex, France
****Laboratoire d'Informatique Théorique et Programmation & Greco de Programmation du C.N.R.S., Université Paris VI, 4 Place Jussieu, 75252 Paris Cedex 05, France

ABSTRACT

A Technological DataBase (T.D.B.) records all the values taken by the physical on-board parameters of a satellite since launch time. The amount of temporal data is very large (about 15 Gbytes for the satellite TDF1) and an efficient system must allow users to have a fast access to any value. This paper presents a new solution for T.D.B. management. The main feature of our new approach is the use of lossless data compression methods. Several parametrizable data compression algorithms based on substitution, relative difference and run-length encoding are available. Each of them is dedicated to a specific type of variation of the parameters' values. For each parameter, an analysis of stability is performed at decommutation time, and then the best method is chosen and run. A prototype intended to process different sorts of satellites has been developed. Its performances are well beyond the requirements and proove that data compression is both time and space efficient. For instance, the amount of data for TDF1 has been reduced to 1.05 Gbytes (compression ratio is 1/13) and access time for a typical query has been reduced from 975 seconds to 14 seconds.

Key Words : data compression, technological database, temporal data.

1 supported by C.N.E.S. and GRECO de Programmation du C.N.R.S.

1. INTRODUCTION

The paper describes a C.N.E.S. study which provides new solutions for the management of technological databases. A Technological DataBase (T.D.B.) contains all the technological telemetry data downlinked from the satellite. Technological telemetry data are values of physical parameters (temperatures, pressures, ...) measured by the on-board platform sensors. They serve to control the satellite, and to follow the performance of on-board equipments. Their storage in a T.D.B. raises two major difficulties :

- as data are produced at a high rate, space needed for storage is very large (for a telediffusion satellite like TDF1, the amount of data exceeds 15 Gbytes at the end of the satellite's lifetime) ;
- in critical cases, when failures occur on board, satellite specialists need efficient access to all data recorded since launch time, and, more generally, users do not need access to one single value, but to a large set of values (e.g., all the values taken by a parameter during one week or one month).

Until now, the design of the T.D.B. was based on the following concept : the telemetry frames are stored in a single file without any transformation. This strategy has two major defaults :

- due to the large amount of data, one has to archive the oldest periods on magnetic tapes ;
- it does not provide efficient access to data.

In 1990, the C.N.E.S. decided to study new solutions for T.D.B. management. The scope of this study was to design and produce a prototype of T.D.B. fullfilling the following requirements :

- the prototype would be able to adapt to different satellites' telemetries and to process real data from geostationary (e.g. TDF1) or polar orbit satellites (e.g. SPOT1) ; so, it would be based on the usage of dictionaries which describe the telemetry format ;
- it would implement data compression methods which guarantee perfect restitution of the information (no bit lost) and cover different types of parameter evolution (some very irregular, others very stable) and different categories of parameter characteristics (length, frequency) ;
- it would provide fast access to all data ;
- it would detect and manage the lack of telemetry lines due to transmission failures.

This prototype has been successfully developed in collaboration with GRECO Informatique du C.N.R.S. and its performances are well beyond the requirements. For instance, the total amount of technological data has been reduced from 15 Gb. to 1.05 Gb. thanks to lossless data compression methods. Extraction of one parameter over 2 weeks (150000 values) which is 975 seconds long in a "traditional" T.D.B. on a CDC 990, is only 14 seconds long with the prototype running on SUN 4/330.

The major principles of our software architecture are as follows. Before storing the final values in the T.D.B., the telemetry frames are decommutated and data corresponding to a parameter are collected on a one-day (or one-pass) basis. Then, for each parameter, a dynamic choice of the most appropriate compression method is made, depending on the stability of the parameter's values. These values are compressed and stored in a specific file (dedicated to the parameter considered) with one block per processed day (or pass). So, during the extraction phase for each requested parameter, we have only to select the blocks coresponding to the requested period by means of a global index, and to decompress these ones.

Section 2 explains how temporal data are represented in our data organization. Section 3 gives a general description of the lossless data compression methods, and section 4 focuses on the specific compression techniques which have been implemented here. Physical organization and software processing are described in section 5. Finally, experimental results are provided in section 6.

2. TIME MANAGEMENT

Parameters have a specific bit-length and frequency. They are measured in a synchronous way. Values are temporal data sent into fixed structured packages called telemetry frames. Frames are composed by telemetry lines which have a regular frequency (one second). All the data of one line are measured at the same time. Transmission of telemetry lines is all the day long for geostationary satellites, but it is gathered in passes for polar orbit satellites (e.g., 8-10 passes per day of 10-12 minutes for SPOT1).

To be efficient, temporal data must first be grouped by parameter : *decommutation* of telemetry frames is a basic principle. For any processed period, it produces a set of values for each parameter. The date of a value can be expressed as a function of the initial date, of the parameter's frequency, and of the range of the value in its set. This strategy leads us to an all-parameter implicit time representation, but we must deal with missing telemetry lines.

The lack of lines can be located and its duration calculated. Rather than replacing a missing value by a special code (this is not easy in particular for one-bit parameters nor is it space efficient as encoding a lack of lines is repeated for any affected parameter), we build a descriptor of the processed period like this :

number of correct line(s), number of missing line(s), number of correct line(s), number of missing line(s) ...

Such a descriptor enables us to find the date of any telemetry line and thus of any parameter value. Hence, decommutation produces a time-

ordered sequence of values (where missing ones do not appear) for every parameter and an all-parameter time description.

3. LOSSLESS DATA COMPRESSION METHODS

We are only interested in lossless data compression methods, also called text compression methods. The reduction in size of the source data is obtained by special encodings applied on them. A corresponding decoding phase is afterwards applied, when needed, to restore the original data. The lossless feature remains to say that the two phases, encoding and decoding, applied in that order to any input makes absolutely no change on the data.

The main interest for compression methods is both for file archiving, and for transmitting data. It is usually thought that compressing slows down the access to data. This is not entirely true in telecommunications because this somehow depends on the speed of the channel relatively to the time spent in encoding and decoding the data. The success of Facsimile machines is likely to be credited to text compression. The present article still provides another example which shows that both space and access time can be improved simultaneously.

It is possible to classify text compression methods according to their probable efficiency. But the performance also depends on the data. More precisely, it depends on the *entropy* of the source, which tells us how regular are the data. If the data are random, the entropy is high, and no method can compress them efficiently.

In this section we describe general data compression methods. Since methods are general, they are purely syntactic. No semantic data compression method is considered. So, compression ratios, which are *ratios between the volume of compressed data and the original volume*, must be appreciated under that condition. Methods commonly save about 50% memory space.

Text compression methods are mainly based on substitutions. It can be considered that the source s, and the encoded text c, are strings over the alphabet {0, 1}. The source s is often itself the encoding of a text written over some more natural alphabet. For instance, s can be the concatenation of ASCII codewords if it is the translation of a text in natural language. We call t the text form which s is translated, and A the alphabet of t. The translation is then described by a function f, which gives the (unique) translation of each letter. Then, defining an encoding remains to specify another function h from A to $\{0, 1\}^*$. The encoding of s is then the translation of t by the function h. The set $\{(f(a), h(a)) / a \in A\}$, or simply $\{(a, h(a)) / a \in A\}$ if function f is implicit, is called the *dictionary* of the compression method. Functions f and h extend to morphisms from A^* to $\{0, 1\}^*$. The lossless condition is satisfied if h, as a morphism, is one-to-one, which means that the set $\{h(a) / a \in A\}$ is a *(variable length) code* (Ref. 1).

The pair of functions, (f, h), leads to a classification of data compression methods based on substitutions. We get four principal classes considering both whether f is uniform (i.e. when all images of letters are words of the same length) or not, and whether the dictionary is fixed or computed during the compression process. Most elementary methods do not actually use any dictionary.

	uniform morphism	non uniform
	relative or topographic encoding	repetition encoding
fixed dictionary	statistical encoding (Huffman)	factor encoding
evolutive dictionary	sequential statistical encoding (Faller and Gallager)	Ziv and Lempel's algorithm

The first line of the above array mentions elementary methods. For suitable data they can give excellent compression ratios. These methods have been eventually chosen to encode the telemetry data.

Statistical encodings are among the most popular methods. The Unix (system V) command "pack" implement the famous

Huffman's algorithm (Ref. 2). It admits a sequential version, discovered independently by Faller (Ref. 3) and Gallager (Ref. 4), well suited for communication, and implemented by the "compact" command of Unix (BSD 4.2). Tests with these methods applied to telemetry data has shown that the process is too general to compress them efficiently.

Finally the last method we have tried is factor encoding. It contains the sequential algorithm of Ziv and Lempel (Ref. 5) on which the "compress" command of Unix (BSD 4.2) is based. This method often gives the best compression ratio on ordinary data. It served as a reference to evaluate the entropy of the sources.

An extensive presentation of text compression methods can be found in Ref. 6.

4. DATA COMPRESSION TECHNIQUES

We have used four different data compression techniques DCT1, DCT2, DCT3 and DCT4. These techniques are adaptative methods based on substitution encoding, relative encoding and run-length encoding. They require data preprocessing in order to optimize the choices of certain parameters in our algorithms. In practice, in order to save time, we have incorporated this analysis into the decommutation process. That is, the data is first analyzed to find which of our compression schemes and which values of the parameters will produce the best compression ratio, and are then compressed using this best method. We now describe in detail our four data compression techniques.

4.1. DCT1

Some parameters only have a very restricted number of values (say 12 for instance). It is therefore possible to assign a special 4-bit code for these 12 values. If there were only 6 different values, one would have used a 3-bit encoding. Thus, the number of bits used for this uniform coding is a parameter of this method. Note that it is however necessary to have a special code to indicate exceptional values. For instance, if the 6 normal values are 121, 125, 129, 133, 137 and 141, one would use the following coding

$121 \rightarrow 000$ $125 \rightarrow 001$ $129 \rightarrow 010$
$133 \rightarrow 011$ $137 \rightarrow 100$ $141 \rightarrow 101$

Here, 110 is unused and 111 can indicate that the next n bits represent an exceptional value. Of course, the dictionary should be memorized. This method can be completed by a run-length algorithm.

4.2. DCT2

This method is a simple relative encoding. For instance, let

117 121 122 117 121 128 121

be a sequence of successive values of a certain 8-bit parameter. By relative encoding, each value other than the first is coded with the relative difference between it and the preceding value. In our example, we obtain

117 +4 +1 -5 +4 +7 -7

The trick is now to use an optimal bit representation to code the differences. In our example, the extremal relative differences are - 7 and 7, which gives an interval of 15 possible values. It is therefore possible to code all the differences by a 4-bit representation. For instance, one can use the leftmost of the four bits to code the sign (1 for + and 0 for -) and the three remaining bits to code the absolute value of the difference. This gives the following code

$0 \rightarrow 0000$ $1 \rightarrow 1001$ $-1 \rightarrow 0001$
$2 \rightarrow 1010$ $-2 \rightarrow 0010$ $3 \rightarrow 1011$ $-3 \rightarrow 0011$
$4 \rightarrow 1100$ $-4 \rightarrow 0100$ $5 \rightarrow 1101$ $-5 \rightarrow 0101$
$6 \rightarrow 1110$ $-6 \rightarrow 0110$ $7 \rightarrow 1111$ $-7 \rightarrow 0111$

Now, since the 8-bit representation of 117 is 01110101, we obtain the following encoding of our sequence

01110101 1100 1001 0101 1100 1111 0111
117 +4 +1 -5 +4 +7 -7

with a compression ratio of $4/7 \cong 57\%...$

If the extremal relative differences were for instance - 2 and + 3, we would have coded these differences by a 3-bit representation only. Determination of the optimal representation

takes place during data preprocessing mentionned above.

DCT2 is especially efficient on data with few repetitions but small relative differences.

4.3. DCT3

This third method is a run-length encoding used for very regular data, that is, data containing a large proportion of repeated values and relative differences almost always equal to 1 ot -1. We describe the algorithm with an example. Consider the following sequence of values of an 8-bit parameter :

90 90 90 91 91 91 91 91 91 91 91 91 91 91 91
91 91 91 90 90 90 91 91 91 91 91 91 91 91 91
91 91 91 91 91 91 91 91 91 91 91 91 91 91 91
91 91 91 91 91 91 91

This sequence can be synthesized as follows

90 (x 3), 91 (x 15), 90 (x 3), 91 (x 31)

or, after relative encoding, by

90 (x 3), +1 (x 15), -1 (x 3), +1 (x 31)

Now the coding becomes easy. The first byte codes the first value (90 on our example). The second byte codes the number of repetition of the first value. The next bytes are decomposed as follows : the leftmost bit encodes the relative difference (1 for +1 and 0 for -1) and the other bits encode the number of repetitions. In our example, we obtain

01011010	00000011	10001111	00000011
90	3	+1 (x 15)	-1 (x 3)

10011111
+1 (x 31)

with a compression ratio of $5/52 \cong 9.6\%$...

There are two obvious problems with this encoding. First, the number of repetitions is limited to 127, the biggest 7-bit number. Second, the coding has to be changed if one of the relative differences is different from 1 or -1. However, it would be convenient to use the same scheme for data containing only exceptional values for which the relative differences exceed one in absolute value.

The first problem can be easily avoided. It suffices to determine the maximal number of repetitions during data preprocessing and code the number of repetitions accordingly. For instance, if the maximal number of repetitions is 56, a 6-bit representation will be enough. If it is 57,243, a 16-bit representation will be required.

Assume now that a n-bit representation is used for the number of repetitions. Then a typical encoding for an 8-bit parameter will have the following form

First value	First number of repetitions
0 / 1	number of repetitions
0 / 1	number of repetitions
0 / 1	number of repetitions ...

on 8 bits + n bits + 1 bit + n bits + 1 bit + n bits + 1 bit + n bits + ...

Since there is at least one repetition, the codes

0	00 ... 0	and	1	00 ... 0

on n bits are never used. This gives the solution to our second problem. If an exceptional relative difference is different from 1 and -1, we use the sequence (on n + 8 + n bits)

0	00 ... 0	relative difference
number of repetitions		

The code | 1 | 00 ... 0 | on n bits is never used, but is saved for future use if the software needs a further development.

4.4. DCT4

Our fourth method combines the ideas of the two previous methods. That is, data is first encoded by relative encoding and then a run-length encoding is applied. As usual, the maximal relative differences and the maximum number of repetitions are determined during the decommutation pass.

Assume for instance that the maximal relative differences are - 3 and + 2 (giving a range of 6 possible values) and that the maximum number of repetitions is 13. Then it is possible to use a 3-bit representation for the differences and a 4-bit representation for the repetitions. Consider for instance the following sequence of values of an 8-bit parameter

90 90 92 92 92 89 91 91 91 91 91 93 93 93 93 93 93 93 93 94 94 94 94 94 94 94 92 92 92 92 92 92 92 92 92 92

By relative encoding, we obtain the sequence

90 0 +2 0 0 -3 +2 0 0 0 0 +2 0 0 0 0 0 0 0 0 +1 0 0 0 0 0 0 -2 0 0 0 0 0 0 0 0 0 0

Then, we can encode the repetitions by run-length encoding. In practice, it is convenient to keep a special code to indicate that we start run-length encoding. Thus, in our example, we would use the following code for the differences

$-3 \rightarrow 110$	$-2 \rightarrow 101$	$-1 \rightarrow 100$
$0 \rightarrow 000$	$+1 \rightarrow 001$	$+2 \rightarrow 010$

and keep the code 111 to indicate that the next four bits will represent the number of repetitions n. Notice that if a value occurs two or three times, it is not efficient to encode the number of repetitions : for instance, for the three consecutive 92, the encoding 010 000 000 will be shorter than 010 111 0011. For this reason, we will only encode repetitions if $n \geq 4$. Consequently, a possible improvement would be to encode n-4 instead of the number n of repetitions. If we don't use this improvement, we obtain for our example the following encoding

01011010 000 010 000 000 110 010 111 0101 010 111 1001 001 111 0111 101 111 1010

which codes, literally

90 0 +2 0 0 -3 +2 (x 5) +2 (x 9) +1 (x 7) -2 (x 10)

leading to a compression ratio of 63 / 296 ≅ 21 %

5. PHYSICAL ORGANIZATION AND SOFTWARE PROCESSING

After decommutation is done, any single-parameter sequence of values is compressed by the chosen specific compression technique. The algorithm number and its parameters are associated with the compressed sequence to create a block. Blocks, whose sizes are variable, are clustered into single-parameter files. For their location, a two-entries index table (the parameter and the processed period numbers) is updated with every parameter's compressed data block adress and size in the file, and then stored in an index file.

The choice of the database updating time unit must be solved with regard to the expected time and space efficiency. In reality, time segmentation must fit the users requirements, knowing that volumes must be sufficient for the compression to be significant, but not too large to avoid the worst cases decreasing general efficiency. It should be noted that, using the previous techniques, updating units are also extraction ones since the entire block must be processed at decompression time. In our experiments, we processed one-day periods for TDF1 and only passes (10-15 minutes) for SPOT1.

In such a system, an any-length multiparameter query is first divided in function of the sub-periods in the database using the time descriptor file, and then is decomposed by parameter. Each block is located using the index file and then read from the parameter file. The sequence of values is decompressed according to the algorithm references set on the block. Finally, extracted values are dated using the time descriptor file and the parameter dictionary.

6. EXPERIMENTAL RESULTS

A modular parameterizable prototype has been developed for TDF1 telemetry (Ref. 7). Another one, intended to process SPOT1 telemetry, was easily built (Ref. 8) from the first one. These prototypes are C programs running on Unix system.

Experimental results presented correspond to programs compiled with the option -O (simple

optimization), and run on SUN4 Sparc 330 for TDF1 and IBM RS6000 520 for SPOT1 (RS6000/520 is 1.5-2 faster than SUN4/330). Given times are user + system in seconds.

6.1. TDF1.

The first table contains compression results in volume and time for one-day sequences of values of some representative parameters. Volumes produced by the Unix "compress" command are also indicated. Considering the produced volumes, DCT1 has not been implemented for TDF1. Remark that any one-day block of values is decompressed in less than one second.

para-meter	origin. volume	DCT1	DCT2	DCT3	DCT4	comp-ress	chosen DCT	comp-ression ratio	comp-ression time	decomp ression time
A301K 8 bits	2696 bytes	1359 b.	678 b.	137 b.	*94 b.*	354 b.	4	3.5%	0.03	0.00
A605G 10 bits	216 Kb.	195 Kb.	69 Kb.	207 Kb.	*69 Kb.*	115 Kb.	4	31.8%	3.59	0.92
A6083 10 bits	216 Kb.	108 Kb.	*73 Kb.*	215 Kb.	74 Kb.	91 Kb.	2	33.7%	2.22	0.86
A673 16 bits	5392 b.	338 b.	679 b.	*6 b.*	8 b.	122 b.	3	0.1%	0.04	0.01
D403H 8 bits	2697 b.	2953 b.	*692 b.*	2702 b.	694 b.	1322 b.	2	25.7%	0.06	0.02
P305K 8 bits	2696 b.	1714 b.	678 b.	*182 b.*	250 b.	554 b.	3	6.8%	0.02	0.00
R101Z 1 bit	5392 b.	_	11 Kb.	_	*7 b.*	123 b.	4	0.1%	0.22	0.04

Database updating was experimented with 50 days of real telemetry (50 days is about 2.5% of the satellite life span estimated at 2000 days). The original one-day volume is nearly 7 Mb. but after compression the average amount of data is 520 Kb. only (add 7 Kb. for management files). Thus, the average compression ratio is approximately 8% (day-ratios go from 7.44% to 9.93%). This would lead to 1 Gb. of compressed data for 2000 days, instead of 14 Gb.

original volume	compressed volume	compression ratio
7.2 Mb.	0.52 Mb.	8 %

Duration of a one-day updating is nearly 6 minutes (3 minutes 1/2 only on IBM RS6000 520).

decommuta-tion	compression	total time
306.1	51.4	357.5

Finally, consider two unfavorable selections of 5 and 20 parameters and note the response times for queries over several period lengths.

period length	decomp-ressed blocks	5 param.	20 param.
one hour	1	4	10
one day	2	11	34
one week	8	51	185

6.2. SPOT1.

Using DCT1 is interesting for compressing some SPOT1 parameters : total volume is decreased by about 20%. We show a few significative examples.

para-meter	original volume	best DCT and ratio	DCT4 ratio
SCCOU3	727 b.	- >100%	76 %
SCECR2	1292 b.	- >100%	47 %
SVROU3	1292 b.	2 49%	27 %
SPHI4	1292 b.	3 66%	27 %
SXTHEX	1292 b.	3 88%	34 %

The next table summarizes time and space results when updating the database with one medium 10-12 minutes length pass.

average original volume	average compres. volume	average compres. ratio	total time
216 Kb.	27 Kb.	13 %	8.4

As a day is composed by 8 passes, one-day amount of data is closed to 220 Kb. instead of 1.75 Mb., and one minute only is necessary to a complete one-day updating. Finally, note that grouping one-day passes and updating the by day would produce an average volume of 199 Kb. and so decrease the volume by about 10%.

7. CONCLUSION

We are currently developing a new operationnal T.D.B. for the satellites TDF1 and TDF2 which is based on this new architecture and linked with a software package for visual data analysis. This new T.B.D. will be an efficient tool to follow the performance of on-board equipments.

8. ACKNOWLEDGEMENTS

The authors would like to thank Michel Mouyssinat whose intuition and enthusiasm led to this fruitful collaboration between C.N.E.S. and GRECO Informatique du C.N.R.S.

9. REFERENCES

1. Berstel, J., and Perrin, D. 1985. *Theory of codes*. Academic Press.

2. Huffman, D.A. 1951. A method for the construction of minimum redundancy codes. In *Proc. IRE* 40 , 1098-1101.

3. Faller, N. 1973. An adaptive system for data compression. In *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*, 593-597.

4. Gallager, R.G. 1978. Variations on a theme by Huffman. In *I.E.E.E. Trans. Inform. Theory* IT 24, 6 , 668-674.

5. Ziv, J., and Lempel, A. 1978. Compression of Individual Sequences via Variable-rate Coding. In *I.E.E.E. Trans. Inform. Theory* IT 24, 5 , 530-536.

6. Bell, T.C., Cleary, J.G., and Witten, I.H. 1990. *Text compression*. Prentice Hall.

7. Arcangeli, J.-P., Crochemore, M., and Pin, J.-E. 1991. Rapports d'études préalables, de conception, d'évaluation et manuels d'installation et d'utilisation du prototype de banque de données technologiques TDF1. Centre National d'Etudes Spatiales, Toulouse, France.

8. Roux, L. 1992. Prototype de banque de données technologiques SPOT1. Université P. Sabatier et Centre National d'Etudes Spatiales, Toulouse, France.