1980-01

Format Based Data Compression

Lyons, Norman R.

Monterey, CA; Naval Postgraduate School

Working Paper Series

DEPARTMENT OF ADMINISTRATIVE SCIENCES

NAVAL POSTGRADUATE SCHOOL

Monterey, California 93940

WORKING PAPER SERIES

The purpose of these working papers is to facilitate dialogue on topics of interest to researchers and practitioners in the broad field of management.

The series provides a way to: (1) inform a variety of persons and communities within academia, industry and government of some of the work in process at the Naval Postgraduate School. (2) promote the generation and sharing of ideas that may not be formally publishable or officially sanctioned. (3) reduce the lengthy period between the submission of an article and its availability for wider readership; and (4) disseminate within the Navy and scholarly communities papers of a theoretical, polemical, exploratory or summary nature.

While the working paper series predominantly reflects the work of the faculty at the Naval Postgraduate School, other researchers and practitioners in the field are invited to submit appropriately typed and documented manuscripts for consideration. All manuscripts will be reviewed by at least three of the Department faculty at the Naval Postgraduate School for quality, relevance, and clarity of presentation. Additions and modifications of the distribution list are encouraged. Please send any comments, criticisms or rejoinders directly to the authors of the various papers. Other communications would be welcomed by the editor.

Roger Evered, Editor
Code 54Ev
Department of Administrative Sciences
Naval Postgraduate School
Monterey, California 93940
Phone: (408) 646-2646

FORMAT BASED DATA COMPRESSION

Norman R. Lyons

January 1980 #80.3

One way to increase the amount of information that can be stored on data storage devices is to develop techniques for reducing the redundancy in the data. The obvious first approach to this problem is the development of variable length records which repeat the fields for tables and array only as many times as required. This approach helps considerably, but you still have the problem of redundancy in the fields that make up the record. In most computer systems, the fields in a record are assumed to be fixed length, padded on the right with blanks if they are alphanumeric and padded on the left with zeros if they are numeric. This kind of approach can be fairly wasteful of storage space since field lengths are usually set so that they can accomodate even the longest of the data items that will be placed in the field. For fields holding data like an individual's last name, this usually means setting aside about 20 or 25 character positions even though most people's last names are considerably shorter.

This problem can be handled using some sort of scheme allowing for variable length fields. There are a number of ways to set up a variable length field system. A survey of the possible approaches is given in the introduction to the article by Maxwell and Severance[1]. The use of variable length fields is fairly simple to implement, but the technique is not widely used. Perhaps the best known system making use of data compression to achieve variable length fields is the ADABAS database management system. The vendors of the ADABAS

---

[1]William Maxwell and Dennis Severance, "Comparison of Alternatives for the Representation of Data Item Values in an Information System", pp 121-124.

system say that their users experience a 50 to 80% reduction in file sizes through the use of compression routines that squeeze out extraneous blanks or zeros.[2] These savings sound a bit large, and one wonders if the users were not overly conservative in defining the field lengths within their records. In addition to these standard techniques of compressing filler characters, more elaborate techniques for eliminating redundant characters in a field are possible. Some of these are given in Date[3], but the author is not aware of the use of any of these approaches in a production system.

The literature shows that large savings can result from the use of fairly straightforward data compression techniques. There are probably a number of reasons why these techniques are not more widely used. First of all, they would have to be implemented at the operating system level so that all processors using a particular machine could access the data files without reprogramming. This means that the software would have to be provided by the manufacturer who generally is also the hardware supplier. The economic incentives to push a risky new feature like this are lacking. On the user side, there would be some reluctance to accept compressed data since this would make it more difficult to transfer files across machine lines. The user objections are likely to be overcome by the spread of data base management systems since these systems generally remove control over the internal data format from the user.

One technique that could be used for data compression is that

---

[2]David Kroenke, Database Processing, p. 260.

[3]C. J. Date, An Introduction to Database Systems, pp. 41-42.

of changing the way in which the data is coded. We use fixed length character codes for EBCDIC or ASCII data, and in an information theoretic sense, this implies that all characters in the character set are equally likely. In the absence of any information to the contrary about our program and the purposes for which it will use the data, the equally likely assumption is the only assumption that can be made on general purpose computer systems. Occasionally in COBOL programs, a programmer will specify the packed decimal notation for numbers instead of letting COBOL default to the zoned decimal notation. This results in about a 50% savings in space over zoned notation. But, the programmer must explicitly declare this type of representation for his data, and it is usually done to save CPU time in converting from zoned to packed for purposes of arithmetic.

In dealing with data in a computer program, we usually do have more information about the type of data to be found rather than having to make the simple assumption that all characters in the allowable character set are equally likely.

In a COBOL program, for example, each record has a definition which tells whether the fields in that record are numeric, alphabetic or alphanumeric. By using the information carried along in the format of the program using the data, we can develop data encoding schemes that are more efficient than fixed length ASCII or EBCDIC codes. The data types that we could use in are given in the Figure 1.

Figure 1 - Allowable Data Types

Type	Length in Bits	Purpose
Binary	1	Used for storing single bits to be used as flags
Numeric	4	Used for storing numeric data. The data could be either edited or non-edited numeric.
Alphabetic	5	Used for storing the alphabetic characters and some punctuation symbols.
Alphanumeric	6	Used for storing alphabetic characters, numbers and some punctuation symbols.
Text	7	Used for storing upper and lower case alphabetic characters, numbers and punctuation symbols.
General	8	Used for storing any character allowed on the character set of a machine with an 8 bit code.

To use this type of variable length coding scheme for data, we have to make a few assumptions. First of all, our system should not recognize any word or byte boundaries. The data is to be stored as bit string data. This means in turn that there must exist some type of format or schema for decoding the data.
This assumption is not problem since in data processing languages like COBOL, there is always a record format containing information about the data stored on the file. All that we are doing is altering our coding system to take advantage of the information given by the format. To further compress the data, we will assume that extraneous padding characters (trailing blanks in the case of alphabetic and alphanumeric data and leading zeros in the case of numeric data) will be squeezed out so that the coding system can use variable length fields. The field boundaries will be determined by markers at the end of the field

unless the field is completely filled. In that case, the marker will be omitted and the length specification in the format will give the field boundary. The markers used will be a set of all one bits. Thus, for a numeric field, four 1's indicate the end of a field and so on. When the data is read into a COBOL record area, it can be re-expanded to the desired fixed lengths for the data and used in the same way that normal COBOL data would be used.

Figure 2 gives the values for each of the characters in each of the types of codes used by this coding system. Each of the types of codes has a different length, and the binary code consisting of all ones is reserved as the end of field marker for each type of code. The exception to this is the binary type code which is only one bit long and does not need any end of field marker. It is assumed that compression will be done on the fields to store them on external I/O devices. All unnecessary filler characters (leading zeros, trailing blanks) will be compressed out before the data is written out on an external device. When the data is called back into a program for use, it can be re-expanded.

Notice in Figure 2 that some of the character codes for numeric data have multiple definitions. This is possible because we can use the format declaration of the data to decide which interpretation is appropriate for a given character. For instance, we will not use the character strings "-", "CR" or "DB" in the same field definition. They are mutually exclusive. Similarly, we will not have occasion to use the characters "$" or "E" together. One will be used with edited numeric data and the other with floating point.

Figure 2 - Internal Representation for Each Coding System.

Numeric Data - 4 bits. Uses codes 0_x to F_x

Character(s)	Value
0 - 9	0 - 9
-, CR or DB	A
$ or E	B
,	C
.	D
*	E
#	F

Alphabetic Data - 5 bits. Uses codes 00_x to $1F_x$

Character	Value
␢	00
A - Z	01 - 1A
.	1B
,	1C
;	1D
-	1E
#	1F

Alphanumeric Data - 6 bits. Uses codes 00_x to $3F_x$

same as BCD code except that 3F is end of field marker

Text Data - 7 bits. Uses codes 00_x to $7F_x$

same as ASCII except that 7F is end of field marker

General Data - 8 bits. Codes 00_x to FF_x

same as EBCDIC except FF is end of field marker

In Figure 2, the character '#' is used to represent the end of field marker. Notice that for numeric values, the symbol "+" is not defined. This is because we can assume all numbers to be positive unless there is a negative sign carried along. The format specifications for the number can tell us how the number is to be printed. The numeric code definitions are set up so that we can use four bits to define either edited or non-edited numeric values. One usually does not wish to store edited numeric data, but it is useful to have that capability built into the system.

It was assumed that alphabetic type data would generally be used for storing proper names. So, besides the 26 alphabetic characters and a blank, we have the ".",",", "-" and "'" to handle most of the situations that will come up in proper names. For the other data types, alphanumeric, text and general, we can use codes already in existence, namely BCD, ASCII and EBCDIC with the exception that a string of all one bits represents the end of field marker.

In converting the data to this internal representation, we ignore all conventional word or byte boundaries. All data is assumed to be bit string and can continue across byte, word or record boundaries. The format statements used with the data will help determine how the bitstrings will be interpreted. On external storage, markers will be used to determine the field boundaries, and the data will be expanded once it is used internally in a program. To show how this would work in practice, suppose we had the COBOL record definition given in Figure 3.

Figure 3 - COBOL Time Card Record.

```
01  TIME CARD
     02 SOCIAL-SECURITY                  PICTURE 9(9)
     02 FIRST-NAME                       PICTURE A  (20).
     02 MIDDLE-INIT                      PICTURE A.
     02 LAST NAME                        PICTURE A(20).
     02 DEPT-NO                          PICTURE 9(5).
     02 HOURLY-CODE                      PICTURE 9.
     02 HOURS                            PICTURE 99V9.
     02 PAY-RATE                         PICTURE 9 (8)V9.
```

Record Length - 69 bytes or 552 bits.

Sample data

Social Security	585019521
Full name	Charles R. Jackson
Department	53621
Hourly code	1
Hours	45 hours
Pay rate	$7.50/hour

When the data in Figure 3 is encoded using the format based technique and extraneous characters are squeezed out, the result is given in Figure 3.

Figure 4 - Time Card Data Encoded Using Format Compression.

Field	Contents	Length (in bits)	Savings due to squeezing extraneous characters (in bits)
SOCIAL-SECURITY	585019521	36	0
FIRST-NAME	CHARLES#	40	55
MID-INIT	R	5	0
LAST-NAME	JACKSON#	40	55
DEPT-NO	53621	20	0
HOURLY-CODE	1	1	0
HOURS	450	12	0
PAY-RATE	750#	16	24
Totals		170 bits	134 bits

Total Savings	69%
Saving From Character squeezing	24%
Savings Due to Data Format	45%

In the example in Figure 4, we get a 69% savings in the total length of the record. Most of this savings comes from using the data format to allow us to encode the data differently, although a substantial portion comes from squeezing out extraneous characters. Notice that when the field is filled to its full length as SOCIAL-SECURITY, MID-INIT, DEPT-NO and HOURS are, no markers are needed to determine the end of the field. The length given in the record description determines the length, and this saves the space that would have been required for markers.

The use of the compression technique on a single record gives an idea of the possibilities, but use in specific applications is needed to determine the savings one can achieve. A great deal depends on the type of data in the file and the extent to which the fields in the file are filled. As a further example, the format based compression techniques was tried on a file containing sample student data for use in a COBOL programming course. The record description for the file is given in Figure 5. The file contains 201 records and each record is 387 bytes long. The results are given below.

Total Length of Original File	77787 bytes
Total Length of Compressed File	34094 bytes
Average Compressed Record Length	170 bytes
Maximum Compressed Record Length	190 bytes
Minimum Compressed Record Length	151 bytes
Savings Due to Squeezing Extraneous Characters	15%
Total Savings	56%

In this example, the savings due to the use of shorter character codes is 41% of the original file length. The actual savings in an application could vary widely from this, depending on the type of data in the original file, the lengths of the fields in the file and the extent to which these fields were filled. The sample file used to generate the data above had fields that were somewhat shorter than might be the case in an actual application file, so the percentage savings generated by squeezing extraneous characters is likely to be somewhat greater in most cases. Still, the savings in file length that can result from the use of format based compression techniques can be very significant.

There are some problems with data compression using a format

Figure 5 - Sample Student Record Description

```
01  MASTER-RECORD.
    02  STUDENT-NO                          PICTURE X(10).
    02  LAST-NAME                           PICTURE X(20).
    02  FIRST-NAME                          PICTURE X(20).
    02  INIT                                PICTURE X.
    02  STREET                              PICTURE X(20).
    02  CITY                                PICTURE X(20).
    02  STATE                               PICTURE XX.
    02  ZIP                                 PICTURE X(5).
    02  PHONE                               PICTURE X(7).
    02  SEX                                 PICTURE X.
    02  AGE                                 PICTURE 99.
    02  BIRTHDAY
       03 MM                                PICTURE 99.
       03 DD                                PICTURE 99.
       03 YY                                PICTURE 99.
    02  COLLEGE                             PICTURE X(20).
    02  MAJOR                               PICTURE X(20).
    02  ADVISOR                             PICTURE X(30).
    02  CREDITS                             PICTURE 999.
    02  AVERAGE                             PICTURE 99V999.
    02  COURSE OCCURS 5 TIMES.
       03  NUMBR                            PICTURE X(6).
       03  NAME                             PICTURE X(20).
       03  HOURS                            PICTURE 99.
       03  P-F                              PICTURE X.
       03  GRADE                            PICTURE 9V99.
    02  ROOM-BAL                            PICTURE 99999V99.
    02  TUITION-BAL                         PICTURE 99999V99.
    02  FEE-BAL                             PICTURE 99999V99.
    02  OTHER-BAL                           PICTURE 99999V99.
    02  TOTAL-BAL                           PICTURE 99999V99.
```

based approach. First of all, there is the additional machine time required to compress and decompress the characters stored on the file. If this were done by the main CPU of the machine, then there might be some question as to whether the savings in I/O device space and data transmission time justified the expenditure of the CPU time. But there are other ways to handle this problem rather than having the CPU perform the data handling tasks. As we move into new generations of machines, we should begin looking for ways to exploit the advances in micropressor technology to build machines that are networks of cooperating microprocessors rather than single stand-alone machines. The possibilities for this type of architecture are especially interesting when combined with developments in database technology. Large database problems are more efficiently handled by networks of cooperating machines rather than single machines, and with the advent of cheap microprocessors, we can make the machines in our network much more specialized. Data compression could be one of the tasks for a microprocessor in a database machine network.

A second problem with compressed data is the issue of transportability and compatability of data across systems. Unless users adopted the same type of compression standards and codes for their databases, it could be very difficult to change data sets across machine lines unless they were exchanged in some standard decompressed code. But this objection is likely to become less relevant with the increasing use of database systems in which the user has little control over the internal format of his data. The potential savings in storage capacity that can result from this data compression technique make it worth using on a production system.

jbb

References

1. Date, C. J., An Introduction to Database Systems. Second Edition, Addison-Wesley Publishing Co., Inc., Reasind, MA, 1977

2. Kroenke, David, Database Processing, Science Research Associates, Chicago, IL, 1977.

3. Maxwell, William and Dennis Severance, "Comparison of Alternatives for the Representation of Data Item Values in an Information System", Database, Vol. 5, Numbers 2, 3 and 4, Winter 1973, pp. 121-139.

PREVIOUS WORKING PAPER TITLES: NEW SERIES ONLY

#.	TITLES	AUTHOR(S)	DATE
80.1	BUDGET ALLOCATION AND ENLISTMENT PREDICTION MODELS OF RICHARD C. MOREY: A BRIEF REVIEW	Dan. C. Boger Kneale T. Marshall	Jan' 1980
80.2	REVIEW OF SCHENDEL & HOFER'S BOOK, <u>STRATEGIC MANAGEMENT: A NEW VIEW OF BUSINESS POLICY AND PLANNING</u>	Roger Evered	Jan' 1980

NAVAL POSTGRADUATE SCHOOL
Faculty of the Department of Administrative Sciences

James Arima	Ph.D.	Northwestern	AP
*Robert Bobulinski	M.S.	Naval Postgraduate School	I
Dan Boger	Ph.D.	U. C. Berkeley	aP
*Phillip Butler	Ph.D.	U. C. San Diego	I
Paul Carrick	Ph.D.	U. C. Berkeley	AP
W. Howard Church	M.S.	U. Southern California	P
John W. Creighton	Ph.D.	U. Michigan	P
*Robert Cunningham	M.S.	Naval Postgraduate School	I
Leslie Darbyshire	Ph.D.	U. Washington	P
Phillip Ein-Dor	Ph.D.	Carnegie-Mellon	Adj
Richard Elster	Ph.D.	U. Minnesota	P
Carson Eoyang	Ph.D.	Stanford	AP
Ken Euske	D.B.A.	Arizona State	aP
Roger Evered	Ph.D.	UCLA	AP
*Edwin Fincke	M.S.	Naval Postgraduate School	I
James Fremgen	D.B.A.	Indiana	P
Reuben Harris	Ph.D.	Stanford	AP
Fenn Horton	Ph.D.	Claremont	AP
*Jerry Horton	M.S.	Naval Postgraduate School	I
Carl Jones, Chairman	Ph.D.	Claremont	P
Melvin Kline	Ph.D.	UCLA	P
David Lamm	D.B.A.	George Washington U.	aP
Shu Liao	Ph.D.	Illinois	AP
Meryl Louis	Ph.D.	UCLA	aP
Norman Lyons	Ph.D.	Carnegie-Mellon	AP
Richard McGonigal	Ph.D.	Michigan State	AP
Alan McMasters	Ph.D.	U. C. Berkeley	AP
Robert Nickerson	Ph.D.	Stanford	aP
*James O'Hare	M.S.	Naval Postgraduate School	I
Clair Peterson	Ph.D.	MIT	AP
Denise Rousseau	Ph.D.	U. C. Berkeley	aP
Robert Sagehorn	M.S.	Naval Postgraduate School	I
Norman Schneidewind	D.B.A.	U. Southern California	P
John Senger	Ph.D.	Illinois	P
*Walter Skierkowski	M.S.	U. Nebraska	I
George Thomas	Ph.D.	Purdue	Adj
Roger Weissinger-Baylon	Ph.D.	Stanford	AP
Ron Weitzman	Ph.D.	Princeton	AP
David Whipple	Ph.D.	U. Kansas	AP
Chester Wright	M.S.	UCLA	aP

<u>KEY</u>::

P	Full Professors
AP	Associate Professors
aP	Assistant Professors
Adj	Adjunct Professors
I	Instructors
*	Persons with Military Rank