LBL-15824
c.2

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

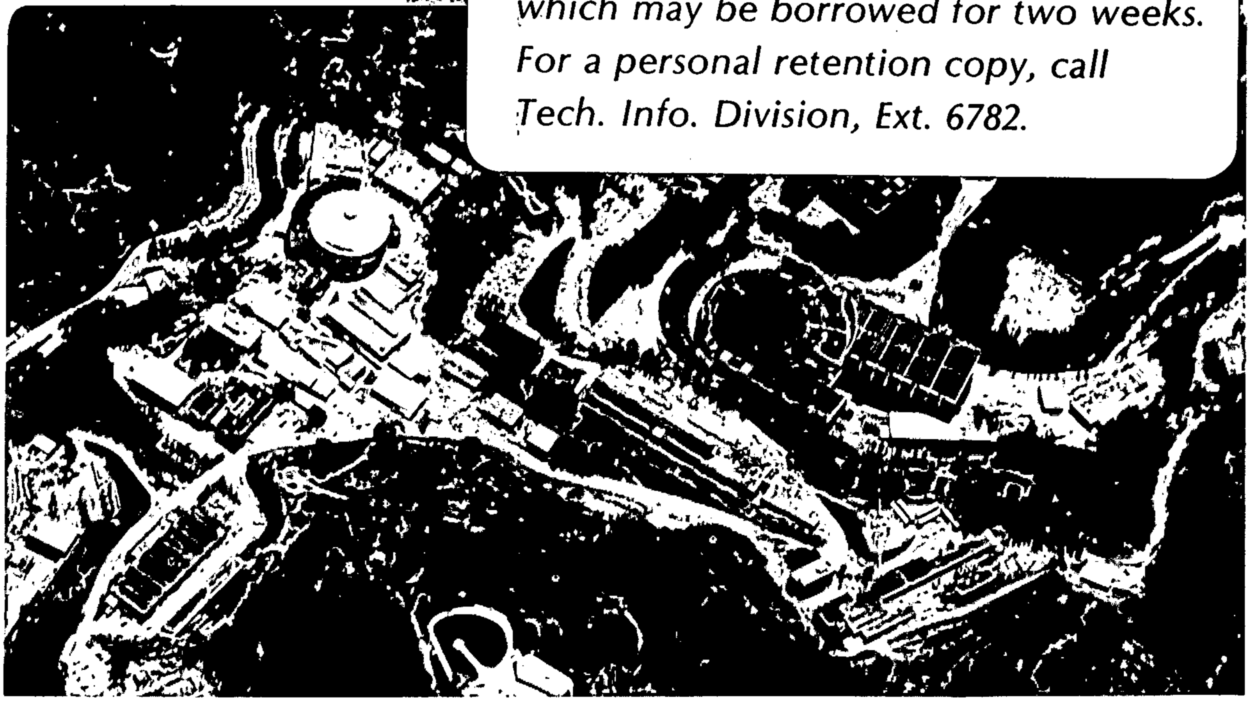
Computing Division

Presented at the Second International Workshop on Statistical Database Management, Los Altos, CA, September 27-29, 1983; and published in the Proceedings

COMPUTER-INDEPENDENT DATA COMPRESSION FOR LARGE STATISTICAL DATABASES

F. Gey, J.L. McCarthy, D. Merrill, and H. Holmes

March 1983



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

LBL-15824
c.2

LBL-15824

COMPUTER-INDEPENDENT DATA COMPRESSION FOR LARGE STATISTICAL DATABASES

Fredric Gey, John L. McCarthy, Deane Merrill, Harvard Holmes

Computer Science and Mathematics Department
University of California
Lawrence Berkeley Laboratory
Berkeley, CA 94720

March 1983

Published in the Proceedings of the Second International Workshop on Statistical Database Management, September 27-29, 1983 Los Altos, CA.

LBL-15824

Computer-Independent Data Compression for Large Statistical Databases

Fredric Gey, John L. McCarthy, Deane Merrill, Harvard Holmes

Computer Science and Mathematics Department
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Abstract

This paper describes a dictionary-driven, hardware-independent data compression scheme for archival storage of large statistical databases. It discusses motivations for this development, storage format requirements, implementation details, access considerations, and possible extensions of the technique. It also analyzes the degree of compression achieved for different types of statistical data files.

1. Introduction and Motivation

SEEDIS is a research and development project on Social, Economic, Environmental, and Demographic Information Systems [MCCA82C], [GEY 81]. The project was initiated in the early 1970's to provide quick, low cost access to databases including the 1970 U. S. Census -- 1.6 billion individual data values for some 400,000 geographic areas. Over the past decade, SEEDIS has evolved from a batch-processing system on Control Data Corporation (CDC) computers to an interactive system on a distributed network of Digital Equipment Corporation VAX 11/780 computers running the DECNET communication system.

By 1977, SEEDIS databases contained nearly 25 billion characters of information, primarily 1970 census data for numerous geographic summary levels, including states, counties, census tracts, enumeration districts, and others. Data were stored in a variety of physical formats, usually a different format for each major data set. Some had been converted to binary representation on the CDC 6000-7000 computers at Lawrence Berkeley Laboratory (LBL). Others had been converted to the CDC display code character set (a 64 character alphabet, with ten six-bit characters per computer word).

Data were stored, for the most part, on an unusual mass storage device, the IBM 1360 photodigital chip store [GEY 75]. In 1977, IBM announced it would no longer maintain this storage device after October 1, 1979, and LBL undertook to search for a replacement. The LBL Computer Center eventually settled on a Cal Comp automatic tape library (ATL), with a 3300 tape reel capacity, robot-assisted tape retrieval, and mounting without operator intervention.

The SEEDIS project was faced with re-archiving its entire database on the new mass storage device. The actual conversion effort consumed 2.5 staff years over a period of 20 months. Given the magnitude of this conversion effort for existing archived data, the project decided to develop a standard archival format in order to minimize the cost of current and future reconversions which might be necessary:

2. Storage Format Requirements

As the project reviewed its long-term archival storage requirements, several common characteristics of statistical databases helped narrow the storage format design goals. First, SEEDIS databases were archival rather than transactional; updates were infrequent and limited in scope. Second, large new databases were continually being added, thus requiring ever-increasing amounts of storage space. Given these basic parameters, several basic design goals were developed, as follows:

- Computer-independent, binary physical storage format for multiple data types
- Dictionary-driven data definition files for data specification and access
- Data compression for efficient storage, retrieval, and transmission

2.1. Computer-Independent Data Formats

For large archival databases such as those in SEEDIS, the data may have a much longer life than the hardware and software used to store and manage them. Since conversion is an expensive and time-consuming process, data need to be stored in formats that do not have

This work was supported by the Office of Health and Environmental Research and the Office of Basic Energy Sciences of the U.S. Department of Energy under Contract DE-AC03-763F00098; and the Department of Labor, Employment and Training Administration under Interagency Agreement No. 06-2063-36.

to be changed as hardware and software change over time.

Unfortunately, most simple standard formats (e.g., fixed length ASCII records) are too inefficient for large numeric databases. Fixed length records require more storage space, more disk accesses for retrieval, and more overhead for data transmission. Furthermore, numeric data represented as characters must be converted to binary before the data can be used in calculations.

In order to provide a format that was simple, efficient, and computer-independent, SEEDIS staff developed a binary storage scheme based on a "virtual machine" for portability of data [HEAL 78]. The commonality of characteristics which constitute this "virtual machine" are as follows:

- storage is divided into 8-bit byte segments
- character (or string) data are stored with ASCII encoding

2.2. Dictionary-Driven Data Definition

Machine-readable data require data specifications that retrieval and applications programs can use. They also require codebooks that human beings can read. The SEEDIS project decided to meet both these requirements simultaneously, by developing a data definition language that could be used to describe both compressed data files and their uncompressed, fixed-length ASCII equivalents.

The basic approach is similar to that of data dictionaries in other database systems and statistical packages. Each self-describing dataset consists of two logical components -- a data definition file (DDF) and a data file (DF). The logical data view is that of a table (or flat file) with a fixed number of rows and columns. Data are arranged so each row of the table contains all the attributes (data elements or columns) of a named entity (e.g., Alameda County), as well as a row label or stub plus any keys necessary for data access and matching. The number of rows is equal to the number of entities in the data file, and the number of columns is equal to the number of data elements in each row.

The basic structure of meta-data elements in the DDF is:

<keyword> = <value(s)>

with one "keyword=value" pair per unit record (line). Keywords occurring before the first data element definition have global effect. That is, they hold for all data elements, unless specifically overridden by keyword definitions within the local environment of a data element definition.

SEEDIS COMPRESSED files, which will be described in Section 3, separate data and description into two distinct physical files. The data file (DF) is in a binary format, while the data definition file (DDF) is in human-readable ASCII representation. One DDF can describe an unlimited number of compressed data files.

Another similar data format known as CODATA (COmmon DATA Format) was also developed by SEEDIS staff [MERR81], [MERR82]. CODATA files contain both data definition and an uncompressed, fixed-length, ASCII representation of the data in a single physical file. These self-describing CODATA files are used to communicate between various independent modules within the SEEDIS system (retrieval, analysis, data entry, graphic display, data export). A software library is available to translate COMPRESSED datasets to CODATA files and vice versa, as well as to extract particular pieces of meta-data information from data definition files.

2.3. Data Compression

Since the volume of SEEDIS databases was quite large (over 25 billion characters of information in 1977 and growing), it was clear that compression techniques were required to minimize costs of data storage, retrieval, and transmission. Initial experimentation revolved around a modified form of packed-decimal format, a data compression technique commonly used by COBOL programmers. This technique, however, was soon rejected because it would save only a fixed amount of space (approximately 50%) for each numeric data item.

Drawing on previous experience with the data, project staff surmised that more substantial compression could be achieved by exploiting several characteristics common to many of the databases in SEEDIS:

- although fixed data fields on source tapes allow enough space for the maximum possible values, most data values require only one or two digits
- many values (particularly zero and missing data codes) are repeated in sequence

The project staff therefore undertook to develop a format which takes maximum advantage of these characteristics, and yet retains the basic objective of hardware independence. The details of this format are described in the following section.

3. Compressed Format Specifications

The SEEDIS compressed format which was developed to satisfy the preceding requirements is basically a binary coding scheme with variable length records. Each data value is stored as a variable-length sequence of 8-bit bytes, preceded by an initial byte containing a 4-bit type code and a 4-bit length count. The format currently provides for three basic types of data: integers, floating point numbers, and character strings. Specific formats for each are described in the subsections below.

3.1. Integer and Fixed Decimal Numbers

Both integers and fixed point decimal data values are handled by a single compressed data storage format. The only difference is in the data definition file, where the "type" for a data element (field) may be defined as either decimal or integer, and a scale factor may be included for decimal data. If a scale factor is indicated, that constant value is multiplied by the stored data value at retrieval time. Scale factors may have any positive or negative value; they may be used to convert units (for example feet to meters) at the time data are retrieved from archived files.

For integer and fixed point decimal data values (type = i), the first four bits of the initial byte contain a code indicating type "i," and the second four bits specify a "length count," the number of bytes required to store the integer in its signed binary representation. The initial byte is followed by the "length count" number of bytes containing the signed binary representation of the integer data value. The integer value 0 (zero) is stored in a special way in the initial byte itself, with a "length count" of zero. Exhibit 1 presents a schematic representation of this compression scheme for integers and values of zero. In this schematic representation, the initial byte is shown as byte position "I", while the variable number of succeeding bytes are labeled "1" through "N". Thus the total number of bytes required for a non zero integer data value is N + 1, while a zero value requires only a single byte of storage.

Exhibit 1: Integer Compression

```
 I| 1  2  3 ...  N|       byte position
--+--+--+--+...+--+
iN| integer value |       contents
--+--+--+--+...+--+

 I|        zero value requires only initial byte
--+
i0|        zero stored in length count
--+
```

Since many computers cannot handle integers larger than can be stored in 32 bits, integers larger than that are automatically stored in ASCII character string form. The DDF plus the access software can automatically translate such data to floating point values on retrieval.

3.2. Floating Point Decimal Numbers

Floating point numbers are stored as two successive integers representing the exponent and mantissa. The first four bits of the initial byte of the exponent contain a code type indicating "e," and the second four bits contain a "length count," the number of bytes required to store the exponent in its signed integer binary representation. The initial exponent byte is followed by "length count" bytes containing the value of the exponent. These are followed by the initial byte of the mantissa, with four bits indicating type "m" and four bits for the "length count" of the mantissa. Finally, there are the "length count" bytes containing the value of the mantissa itself. Exhibit 2 summarizes the compression scheme for floating point numbers in schematic form.

Exhibit 2: Floating Point Compression

```
  | 1 ...  P|  | 1  2 ...  Q| byte position
--+--+...+--+--+--+--+...+--+
eP| exponent |mQ| mantissa  | contents
--+--+...+--+--+--+--+...+--+
```

As the schematic representation shows, the amount of storage required for floating point numbers is 2 + P + Q, where P and Q are the number of bytes required to store the integers representing the exponent and mantissa, respectively. The values of P and Q, in turn, are N + 1 and M + 1, respectively, ~~where N is the the number of bytes required~~ to represent the signed integer value of the exponent, and M is the number of bytes required to represent the signed integer value of the mantissa.

This compressed format for floating point numbers is, in general, less efficient than standard 32-bit binary representations. It requires a minimum of four bytes (32 bits), and frequently may require five or six bytes. On the other hand, it provides a single representation for decimal numbers of virtually unlimited precision. Because of different formats and precision limits for single and double precision on 32, 60, and 64 bit machines, and because most SEEDIS data are integers or fixed decimal numbers, the staff decided to trade compression efficiency for computer hardware independence. Depending upon the data and applications, if large amounts of floating point data became the rule rather than the exception, it might be desirable to add more data types to achieve greater compression at the expense of precision.

Scale factors may be used with the floating point decimal format just as with the fixed decimal format.

3.3. Character Strings

For character string data values (type = a), the first four bits of the initial byte contain a code indicating type "a," and the second four bits contain a "length count," the number of bytes required to store the binary integer representation of the *length* of the character string. The initial byte is followed by "length count" bytes containing a binary integer, "S," and then S bytes containing an ASCII representation of the character string itself. Character strings are further compressed by removing trailing blanks.

For example, if we wish to store a 30-byte character field which contains 6 trailing blanks, the initial byte will contain a code for type "a" and a value of 1, indicating that the length of the character string will be stored in the next 1 byte. The second byte will contain a binary integer representation of the value "24," and that will be followed by 24 bytes of the actual character string, represented in ASCII. Exhibit 3 shows a schematic representation of the SEEDIS compressed format for character strings.

Exhibit 3: Character String Compression

```
 I| 1  2 ...  R| 1  2 ...  S|   byte position
--+--+--+...+--+--+--+...+--+
aR| integer "S"|string value|   contents
--+--+--+...+--+--+--+...+--+
```

3.4. Repetition of Data Values

The SEEDIS compression scheme is rounded out by a fourth data "type" which specifies a repeat count (type = r) of the data value which follows it. Repetition of data values is an extremely important consideration in statistical summary data, which often consist of multi-dimensional cross tabulations of microdata files. For example, 1970 Fourth Count census data for each geographic area consist of 1178 data items for each of five race categories (total, white, black, hispanic-origin, other). For a large number of geographic areas, all data for the latter three race groups are zero or suppressed. Thus, for those three racial categories, groups of 1178 data values can be replaced by six bytes of actual storage, assuming the appropriate arrangement has been made for physical contiguity of the data (which sometimes requires transposition of the data matrix). Exhibit 4 presents a schematic representation of the repeat data type, using zero as an example of the repeated value. The first four bits of the initial byte contain a code indicating typle "r," and the second four bits contain a "length count," the number of bytes required to store the binary integer representation of the repeat count W (the number of times the data value is repeated).

Exhibit 4: Repetitive Value Compression

```
 I| 1  2  3 ...  W| I| byte position
--+--+--+--+...+--+--+
rW| repeat count  |i0| contents
--+--+--+--+...+--+--+
```

Since repeated values of zeros and missing data codes occur frequently in scientific and statistical databases, this type of compression is particularly effective. The repeated value need not be zero, of course. It can be any of the currently recognized data types -- integer, floating point, or character string.

4. A Simple Example

Using the basic building blocks outlined above, we illustrate in this section a brief example using fragments of the data definition file and a single abbreviated data record from the 1980 Census Summary Tape File 1 (STF1).

Exhibit 5 shows a slightly simplified version of the global portion of a data definition file plus definitions for several individual tabulations. This data definition file contains information for both the compressed data file and its fixed field ASCII equivalent. The various meta-data elements appearing in Exhibit 5 are fully defined in [MCCA82A]. The meta-data element "POSITION" gives the sequential field position of the data element within each compressed data record, while the corresponding "START" and "LENGTH" meta-data elements give the physical position and field length in the corresponding fixed-length ASCII CODATA file.

Exhibit 5: Example Data Definition File

```
DATABASE = 1980 Census STF1 Fragment
NDE = 17
AREAS = 4
RECORD_LENGTH = 40
TYPE = A
DE = FIPS.STATE
   START = 1
   LENGTH = 2
   POSITION = 1
DE = FIPS.COUNTY80
   START = 4
   LENGTH = 3
   POSITION = 2
DE = STUB.GEO
   START = 8
   LENGTH = 33
   POSITION = 3
DE = TAB3(1)
   TYPE = I
   START = 42
   LENGTH = 9
   POSITION = 4
   UNIVERSE = 100-Percent Count of Persons
   HEADER =#100-Percent Count of Persons#
DE = TAB74
   TYPE = D
   START = 52
   LENGTH = 9
   POSITION = 5
   SCALE = 0.001
   UNIVERSE = Families
   HEADER =#Median Family Income In 1979#
   HEADER =#(in thousands of dollars)#
DE = TAB12
   STRUCTURE = ARRAY
   DIMENSION = RACE(12)
   TYPE = I
   START = 61
   LENGTH = 10
   POSITION = 6
   UNIVERSE = Persons
CATEGORY_SET = RACE(12)
   CATEGORY = White
   CATEGORY = Black
   CATEGORY = American_Indian
   CATEGORY = Eskimo
   CATEGORY = Aleut
   CATEGORY = Japanese
   CATEGORY = Chinese
   CATEGORY = Filipino
   CATEGORY = Korean
   CATEGORY = Asian_Indian
   CATEGORY = Vietnamese
   CATEGORY = Hawaiian
END DDF
```

There are 17 data elements in Exhibit 5, namely FIPS.STATE, FIPS.COUNTY80, ..., TAB74, and the 12 elements of TAB12. Note that the last DE entry, TAB12, is a vector (or one-dimensional matrix) rather than a simple single-valued field. This fact is denoted by the meta-data information "STRUCTURE = ARRAY." Information about the 12 components of TAB12 is located in the category set "RACE(12)" named in the "DIMENSION" statement. This notation considerably simplifies specification of multi-dimensional arrays, which frequently occur in scientific and statistical data. It also saves space and processing time in handling large data definition files. For such array data elements, the "POSITION" given is the position of the first cell in the array. Each element of an array is stored exactly as a simple data element. Retrieval software computes the linearized position number of other cells from a standard array notation such as "TAB51(3,12,4)". Similarly, the "START" for such data elements gives the starting location of the first cell, while "LENGTH" gives the length of each successive cell in the array.

Exhibit 6 shows a fixed format ASCII representation of a single data record as defined by the DDF in exhibit 5. As implied by the "RECORD_LENGTH" global meta-data item of the DDF, each logical record is comprised of five 40-character physical records (lines), with 20 characters of padding at the end of the last line. This is the type of format that constitutes the data portion of a CODATA file, and it typifies formats used for data export by many agencies such as the U.S. Census Bureau. Note how fields are allocated to accommodate the largest possible value that might occur in that field -- even though most of the actual values are much smaller. The indication "SCALE=0.001" under "DE=TAB74" specifices that the stored value (17240), when multiplied by the scale factor (0.001) will yield the median family income in thousands of dollars (17.240).

Exhibit 6: Fixed-length ASCII Data Record

byte position in physical record

```
         10        20        30        40
++++|++++|++++|++++|++++|++++|++++|++++|
06 003 CA ALPINE
       1097     17240       912         0
        169         0         0         0
          0         0         0         0
          0         5
```

Exhibit 7 presents a schematic representation of the compressed form of the data record from Exhibit 6. It shows how 200 bytes of original data compresses to 38 bytes of output data, or 19 percent of the original record size.

Exhibit 7: Compressed Data Record

byte position

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
a1  2  0  6 a1  3  0  0  3 a1  9  C  A     A  L  P  I  N  E
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

byte position (continued)

```
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
i2  1097 i2 17240 i2  912 i0 i2   169 r1  8 i0 i1  5
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

A close comparison of Exhibits 6 and 7 shows that space savings occur in four major ways: (1) truncation of the area name by removing trailing blanks; (2) binary encoding of integer and fixed point decimal data (e.g. one byte will now hold numbers from -127 to +127, two bytes will now hold numbers from -32,767 to +32,767, and these can be scaled by an arbitrary constant in the data definition file for decimal fields); (3) run-length encoding of repeated values; and (4) elimination of padding at the end of the last physical record. The largest single savings comes from the repetition of 8 successive zeros, which originally occupied 80 bytes, but takes only three bytes in compressed form (byte positions 34-36).

The next section provides more extensive empirical data about the degree of compression achieved over large numbers of records for some actual SEEDIS datasets.

5. Analysis of Compression in SEEDIS

Previous sections have described SEEDIS compression methods and an example of how these compression techniques work for a single data record. This section analyzes the results of compression on entire data files and data bases. In general, data from public agencies such as the U. S. Census Bureau are formatted into fixed-size records for each file, regardless of the contents of the data. Thus, for each data base, a single constant number of bytes is associated with each record as received by LBL. Of interest, therefore, are:

- The distribution of sizes of output records after compression
- The ratio of compressed record size to original record size for different databases
- The cumulative percentage of all records in a data base whose compressed size is less than some percent of the original size
- Whether compression correlates with any particular data item contained within the data record

Exhibit 8 shows the distribution of records for a single data base (1980 Census Summary Tape File 3 (STF3), county records) whose compressed size is expressed as a percentage of the original size. For this database (and level of geography) the median compressed record was 25 percent of the size of the original record.

Exhibit 9 summarizes these distributions over several data bases, showing the cumulative percentage of records with compressed size less than some fraction of original size. The best compression is achieved on the 1980 Census Equal Employment Opportunity (EEO) data file, which contains counts of labor force for 514 detailed occupational categories. The high degree of compression is due to the large number of repeating data values of zero in this file. The worst cases are the air quality and mortality datasets, which contain a high proportion of floating point data.

Finally, we conjectured that the compression might be related to the total population of the geographic area corresponding to the data record, since the proportion of data with values of zero or suppression (missing data) codes increases for smaller areas. Exhibit 10 is a scatter plot of compressed record size versus the logarithm (base 10) of 1980 total population for the STF3 data base.

A straight-line fit is a remarkably good one, which is especially significant because it suggests that one can sometimes make quantitative estimates of storage requirements for each record or set of records based upon a single value contained within the data.

Our quantitative exploration of compression results is continuing and we hope to use these results in developing further analytic models of compression.

Exhibit 9: Compression in Selected Data Files

Fraction of Original	Cumulative Percent of County Level Records					
	EEO	STF3	CDB	STF1	AQ	MOR
0 to 5%	58	0	0	0	0	0
5 to 10%	93	0	0	0	0	0
10 to 15%	98	0	0	0	0	0
15 to 20%	100	3	1	0	0	0
20 to 25%	100	43	5	1	2	0
25 to 30%	100	92	94	18	7	0
30 to 35%	100	100	100	87	20	1
35 to 40%	100	100	100	100	35	3
40 to 45%	100	100	100	100	46	8
45 to 50%	100	100	100	100	59	15
50 to 55%	100	100	100	100	73	28
55 to 60%	100	100	100	100	81	49
60 to 65%	100	100	100	100	88	64
65 to 70%	100	100	100	100	92	75
70 to 75%	100	100	100	100	95	85
75 to 80%	100	100	100	100	98	92
80 to 85%	100	100	100	100	99	97
85 to 90%	100	100	100	100	99	100
90 to 95%	100	100	100	100	100	100
95 to 100%	100	100	100	100	100	100
Mean %	5	25	27	32	47	61

EEO -	1980 Census Equal Employment Opportunity
STF3 -	1980 Census Summary Tape File 3
CDB -	1940-1977 City County Data Book
STF1 -	1980 Census Summary Tape File 1
AQ -	1974-1976 Air Quality
MOR -	1968-1972 Age-Adjusted Mortality

Exhibit 8: Size of Compressed Records

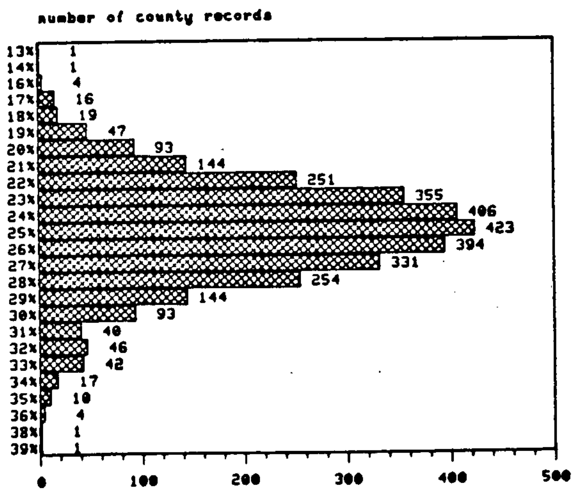
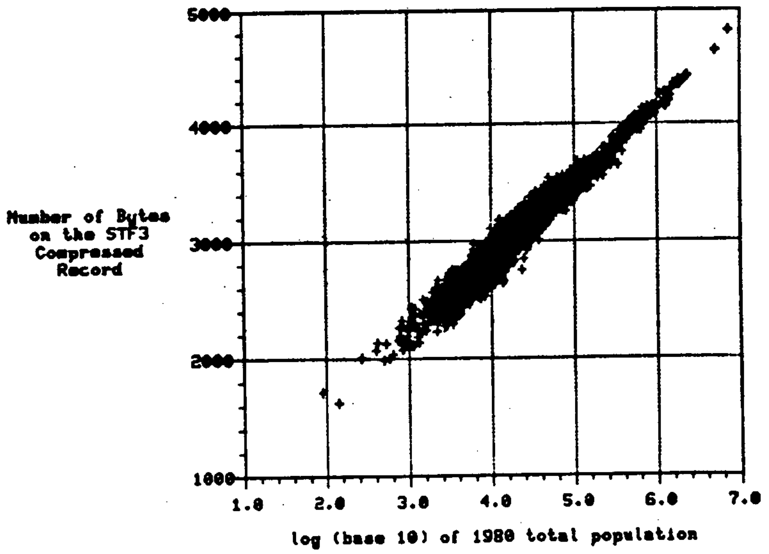


Exhibit 10: Compressed Size vs. Population



6. Access Methods for Compressed Data

In the simplest case, a SEEDIS compressed dataset consists of four physical files. A dataset known as MYFILE, for example, might consist of files having the following names:

MYFILE.DAT: Compressed binary data file (DF)

MYFILE.NDX: Index file containing pointers to individual records (entities) in MYFILE.DAT

MYFILE.DDF: ASCII data definition file (DDF) describing individual named attributes (data elements) in the DF

MYFILE.DDX: Index file containing pointers (sequence numbers) corresponding to each data element in the DDF.

The multi-file scheme permits considerable flexibility in the way data are stored and accessed. For example, large county-level data sets are normally stored with one DAT and its corresponding NDX file for each state, e.g.:

S01.DAT: Alabama DF
S01.NDX: Index for S01.DAT
S02.DAT: Alaska DF
S02.NDX: Index for S02.DAT
MYFILE.DDF: Same as before
MYFILE.DDX: Same as before

Corrections, if necessary, can be easily made to one state at a time. Furthermore, not all states need to be stored on the same disk pack or even on the same node (host computer). Automatic schemes in SEEDIS [MERR83] provide for selective caching of only those files actually required by the user. Because the meta-data (DDF and DDX) are physically separate from the data files (DAT and NDX), meta-data elements (for example data element labels or scale factors) can be conveniently changed without the need to recompress the data.

6.1. Record Access Mechanisms

As stated above, the NDX file provides pointers to the individual records (entities) of the compressed SEEDIS data file (DAT file). In the example above, the file S01.NDX is itself a simple CODATA file containing four data elements for each county in Alabama: the FIPS (Federal Information Processing System) state code, the FIPS county code, the size in bytes of the compressed record for that county, and the physical block location of the start of that county in the file S01.DAT. Now suppose that at some future date the FIPS county numbering scheme for Alabama is changed to accommodate the splitting of a county or the combination of two present counties. Without modifying the file S01.DAT, one can easily modify S01.NDX so that the existing data can be retrieved via the revised county codes. New county codes not corresponding to existing data are simply omitted from the new NDX file, resulting in a missing data indication when data are extracted. Because the NDX files are much smaller than the compressed DAT files, multiple sets of NDX files corresponding to various county definitions can easily be created, all pointing into a single set of large DAT files. Entities (counties) not in the original DAT files can be left missing, or can be created by aggregation or disaggregation and appended to the original DAT file. Each set of NDX files provides pointers to a complete and non-overlapping set of entities, for example 1960 FIPS counties, 1980 FIPS counties, etc., with only a small increase in stored data.

6.2. Intra-Record Access

When retrieving a particular data value from a particular record, the access software must search from the beginning of the record to the particular data value. Thus, although some time can be saved by computing intra-record positions over repeated data items, access time is generally linear, with the last data item in a record requiring the maximum access time. This has not posed severe problems for SEEDIS data records containing up to 1000 data items but it could for much larger data records. The 1980 Census EEO (Equal Employment Opportunity) database, for example, contains over 12,000 items per record. Access time for some data elements will be slow if the current scheme is not enhanced.

Recognition of the limitations of linear access has provided impetus for additional research on compression methods. This research has lead to a variety of general results [EGGE 81]. In the modified approach, which has yet to be incorporated in SEEDIS, all counts are removed from the data file and stored in a separate header. The counts are cumulative, allowing the header to be searched in logarithmic time. The header is used to form the base level of a B-tree index into the data record, which further improves the access time by increasing the rate of the logarithmic search.

An even simpler scheme involving less storage overhead would be to include in the NDX file not only the starting block location of data element 1 in the DAT file record, but also (for example) the starting block location of data element number 1001, data element number 2001, etc. Considerable time would be saved in retrieving from large files like the 1980 Census EEO file.

6.3. Other Access Considerations

A minor problem which arose during software implementation was the fact that most machines store data in contiguous byte sequential format, but DEC equipment (PDP-11 and VAX) stores numeric data in inverted order within each word of the machine. This fact has been noted in articles on portable software [NEAL 78]. Thus the VAX implementation of the access software had to be slightly different from what it might be on IBM or other hardware.

One other compromise to portability was made within the data records themselves. In order to efficiently use the FORTRAN language for accessing variable-length data records on CDC machines, the beginning of each record contains the CDC word count as well as the total length of the record in bytes.

7. Current and Future Developments

During the past two years a major development effort in SEEDIS has been the design of extensions to meta-data structures for more efficient processing of large summary databases [MCCA82A, MCCA82B]. These enhancements include:

- matrix data elements
- category set specifications
- comprehensive handling of missing data

We are currently designing an enhanced compressed interchange file wherein meta-data as well as data will be stored in the same binary compressed format. Compressed files will replace CODATA files as the standard format for internal interchange of data between SEEDIS modules, in order to provide for more efficient data transfer and conversion. This will be an upwardly compatible enhancement to the original compressed data format, adding new compressed data types for specification of arrays, recursive hierarchical structures, multiply occurring data values, multi-valued missing data codes, etc.

8. Conclusions

With minor modifications and compromises, the computer-independent compressed data storage format described above has remained the SEEDIS standard format for six years. We are currently adding the 1980 Census to the archive, bringing it from 3 to 6 billion data values within a three-year period. We are able to access and decompress on VAX computers data which were archived on tape by CDC computers in 1977.

These compression methods have been very successful from the standpoint of reduction in storage space. Most SEEDIS files occupy from twenty to fifty percent of their original space. Compression of integer and fixed decimal fields to variable-length sequences of bytes, and run-length encoding of repeated values, have accounted for the majority of space saved.

Although the methods currently used to access SEEDIS compressed data files have been adequate for retrieval of data for specified geographic areas, they are not efficient for queries based on data values. Analysis indicates that changes in access methods as well as changes in the compression scheme itself could considerably improve performance for such queries.

Work is currently under way to implement additional compression techniques, different access methods, and compressed data types to accommodate meta-data (e.g., data description files) as well as data. The SEEDIS project plans to use the compressed data format as a medium for internal exchange as well as for archival storage, in order to improve data transmission efficiency between application modules.

9. Acknowledgments

Carl Quong, head of the LBL Computer Science and Mathematics Department, is responsible for the research environment in which this work was conducted. Others who made these results possible include Harvard Holmes, who has directed the SEEDIS project and guided design of the compression methodology, and Bob Healey, who contributed to the design and implemented the initial versions of compression. Wayne Graves made substantial contributions to improving the efficiency of the compression programs, and provided ideas for intra-record access enhancements.

10. References

EGGE 81 Eggers, S., Olken, F., and Shoshani, A., "A Compression Technique for Large Statistical Databases," *Proceedings of the Seventh International Conference on Very Large Databases*, Cannes, France, September, 1981, 424-434.

GEY 75 Gey, F. and Mantei, M. "Keyword Access to a Mass Storage Device at the Record Level," *Proceedings of the First International Conference on Very Large Databases*, Framingham, Massachusetts, September, 1975, 572-588.

GEY 81 Gey, F., "A Beginner's Guide to SEEDIS," Lawrence Berkeley Laboratory Report LBL-11198, January, 1981.

HALL 81 Hall, D., Scherrer, D., and Sventek, J., "A Virtual Operating System," *23 Comm. ACM 9*, September, 1980, 495-502.

HEAL 78 Healey, R., "BYTER and DBYTE," Lawrence Berkeley Laboratory, Internal Document, May, 1978.

MCCA 82A McCarthy, J., "Enhancements to the Codata Data Definition Language," Lawrence Berkeley Laboratory, LBL-14083, February, 1982.

MCCA 82B "Metadata Management for Large Statistical Databases," *Proceedings of the Eighth International Conference on Very Large Databases*, Mexico City, September, 1982.

MCCA 82C McCarthy, J. L., et al., "The SEEDIS Project: A Summary Overview," Lawrence Berkeley Laboratory, PUB-424, May, 1982.

MERR 81 Merrill, D., "CODATA Users' Manual," LBID-021, revised October, 1981.

MERR 82 Merrill, D., "CODATA Tools: Portable Software for Managing Self-Describing Data Files," Lawrence Berkeley Laboratory, LBL-15441, *Proceedings of Computer Science and Statistics: Fifteenth Symposium on the Interface;* Houston, Texas, March 16-19, 1983.

MERR 83 Merrill, D., McCarthy, J., Gey, F., and Holmes, H.; "Distributed Data Management in a Minicomputer Network: The Seedis Experience;" *Elsewhere in the proceedings of this workshop*.

NEAL 78 D. Neal and V. Wallentine, "Experiences with the Portability of Concurrent Pascal," *Software Practice and Experience*, V.8, NO. 3, 1978, pp. 341-353 (specifically p.346).