# DATA COMPRESSION TECHNIQUES FOR ECONOMIC PROCESSING OF LARGE COMMERCIAL FILES

James E. Mulford and Richard K. Ridall
Ridall Associates, Inc., Paoli, Pennsylvania

## ABSTRACT

The application of compact coding, differencing and other techniques to indexed sequential files is discussed. The effects on system performance are discussed and reductions of almost 80% in mass storage requirements for a particular file are reported.

KEY WORDS AND PHRASES

data compression, compact coding, file compression, file storage, numeric coding, alphabetic coding, differencing, reduced alphanumeric image coding, statistical code development

## I. Introduction

This paper reports on the practical application of data compression techniques to large commercial files in an IBM System/360 and DOS environment. The result of compressing one file in particular is used to illustrate quantitatively the achievable compression and its impact on the associated system. Perhaps the most significant result of this effort was not the dramatic compression achieved but the fact that it was achieved without materially altering the existing application system or programs to which the compression was applied.

Data compression, or compaction as it is sometimes called, refers simply to translating a representation of a set of data into a smaller one, without loss of information. The practical effect of this transformation is to save channel time in a data communications system or storage space in a file. The process offers potential for cost saving and other benefits which can be traded-off against cost. For example, compression has been used to reduce bandwidth (consequently power, weight and physical size) required for transmitting weather data from satellites to earth. It has also been used to reduce the size of very large files of textual information, as reported by Snyderman and Hunt.(18) It is also applicable to many commercial/industrial files containing a variety of different types of information as illustrated by the file described below.

File compression yields benefits in addition to size and consequent storage cost reduction. Because data is transferred to and from storage media (core, disk, etc.) in compressed form, program execution time is usually reduced even after making allowance for the decompression computational load. The saving in channel and core interference times are large enough in practice to provide savings even in a multiprogramming environment since CPU time is still under-utilized in most commercial applications.

In the next section, the attributes of a particular commercial file are described as background for a discussion of the compression techniques applied to it. The techniques themselves, which required no reprogramming of the original application programs, are presented next. In the last section, the results achieved from file compression are reported and performance is compared with that of the original system.

## II. File and Program Environment

The file under consideration is the Master File of an application system which provides daily a list of records which meet certain search criteria to the users. There are presently only four COBOL and assembly language programs which use the Master File for all system functions including searching and file maintenance.

In uncompressed form, a Master File record is fixed length at 280 bytes, of which 20 bytes may be used as search keys. With 800,000 base records, the file is about 250 million bytes long including overflow, etc. The file is physically stored in IBM's Indexed-Sequential format to achieve the desired efficiency in maintenance and provide random retrieval of the relatively few records that are accessed in the daily processing.

The record data consists of multiple personal names, addresses, certain legal data, amounts of money, dates, and various control codes. Within a given record, fields may or may not be present, and within a field, the data is variable length. The application is required to run under DOS which dictates that this variable length data be partitioned into fixed length records. In the uncompressed file, each base record may have zero to three trailers.

## III. Compression Techniques

The techniques used for compressing the above file for disk storage fall into two groups, Logical or Statistical. They are distinguished by the data characteristic which provides the basis for compression. However, in some cases, more than one technique was applied to a particular datum.

### Logical Techniques

Certain fields within a single record may be logically related, or partly redundant. For example, in our Master File there are three dates describing events which must happen in a given sequence. First, there is a date on which a legal transaction took place. Then the date on which the transaction data were entered into the Master File is recorded. If any corrections are made to a record, the date of the last correction must also be entered.

Not only must the three dates be in ascending order, but no date can be before 1940. Allowing for system life to the year 2000, the range of date fields must be less than 22,000 days. Further, it is very likely that the date of data entry will be within two days of the transaction date, so that its range can be even more restricted if it is referenced to the transaction date. A similar situation exists with the third date so that the difference between dates can be stored rather than the dates themselves. In the first instance, the difference between the actual date and 1940 is stored. Then, only the difference between the first date and the second are stored, and so forth. By representing these difference in binary, a compression from 18 bytes to less than four bytes is achieved.

This differencing technique can be applied to other fields as well. In serial files sequenced by keys such as Social Security, account, purchase order, or part numbers, the application is obvious. Differencing can also be applied to textual data, such as names, and result in a saving.

The situation is analogous to transmission of a digitized photographic (TV or whatever) image. Rather than transmitting (or storing) the absolute value of each point scanned, its difference from the last can be used with an occasional reference point included. This often results in fewer bits of information transmitted because dark and light areas are in "significant" patches rather than random.

In other cases, more special techniques were used which depended on particular relationships between fields. Generally, these special techniques were applied to fields where it was possible to derive one field from one or more others, and for infrequently appearing fields.

### Statistical Techniques

Alphanumeric Coding — Within this class of techniques one in particular was the most important contributor to data compression because it applied to the greatest number of long fields. It is called Reduced Alphanumeric Image (RAI) or compact coding technique. In it, the actual frequency of occurrence for each character (obtained by sampling the file) is used to derive a variable length binary code. To illustrate the process consider the following example:

> The RAI technique is demonstrated by a simple case where the source alphabet consists of only the four letters "A" through "D" and a space character as a separator. The probability that the next character of a string will be a particular one is assumed to be the values shown in Table I.

TABLE I.

| Character, $C_i$ | Probability, $P_i$ |
|---|---|
| Space | 0.4 |
| A | 0.2 |
| B | 0.2 |
| C | 0.1 |
| D | 0.1 |

A string of such characters contains a certain amount of information (E) which may be computed as follows:

$$E = \sum_{i=1}^{i=5} P_i \log_2 \frac{1}{P_i}$$

For the assumed character set, "E" is approximately 2.1 bits. That is, on the average a character conveys 2.1 bits of information. It is generally not possible to achieve this degree of compression with practical codes but it is possible to approach the limit. For example, the binary code shown in Table II will require 2.2 binary digits, on the average.

TABLE II.

| Character | Code | x Probability | = Binary Digits, Average |
|---|---|---|---|
| Space | 1 | 0.4 | 0.4 |
| A | 01 | 0.2 | 0.4 |
| B | 000 | 0.2 | 0.6 |
| C | 0010 | 0.1 | 0.4 |
| D | 0011 | 0.1 | 0.4 |
| | | Total Binary Digits, Average | 2.2 |

There are many sets of codes that are just as compact. Some others, shown along with the first example, are set forth in Table III.

TABLE III.

| Character | $Set_1$ | $Set_2$ | $Set_3$ | $Set_4$ |
|---|---|---|---|---|
| Space | 1 | 0 | 00 | 11 |
| A | 01 | 10 | 01 | 10 |
| B | 000 | 110 | 10 | 01 |
| C | 0010 | 1110 | 110 | 001 |
| D | 0011 | 1111 | 111 | 000 |

Although these code sets are all equivalent in terms of compression achieved, some may be preferred in a particular situation to achieve processing efficiency. $Set_2$, for example, facilitates parsing a binary stream into codes using simple rules. The end of a character is denoted by a zero, or the fourth binary digit, whichever is first.

Even greater compression can be achieved if the frequency of occurrence of character pairs is also considered. To illustrate, if the letters "B" and "C" occur the same number of times as they do in the above example, but "C" always immediately follows "B", then the same character set can be coded in an average of 2.1 binary digits instead of 2.2 by including the character "BC" and eliminating the character "C".

This somewhat more advanced approach was, in fact, used for certain pairs in name fields. The format for names in both the original and compressed Master File was:

Last, first e.

Due to the format, the two pairs ",b" and ".#" occur frequently where the "b" and "#" symbols represent blanks and end-of-field respectively. It was found advantageous to code these and 10 other pairs as unique "characters". For the actual character, frequencies, probabilities and codes used, see Exhibit A1. Using this code, an average of 4.14 binary digits are required to encode one source character from a set of 43 characters. The letter frequencies found for names differ significantly from the frequencies reported previously in english text by Nugent and Vegh (13).

Character frequencies for address data differ significantly from those of name data and a different code set was used. Some of the differences are to be expected because street addresses are usually in the format:

nnn streetname st.

where "n" is a decimal digit between 0 and 9. Note that a sequence of 2, 3, or 4 digits followed by a space or N,E,S or W would be common, so that a code prefix subset and a teletype-like shift character would provide quite compact coding. Note also that the last four characters "bst." could be expected to occur frequently and so code efficiency could be improved by coding the sequence as a single character. These expectations were verified by a sample count on address fields and the resulting code set is shown in Exhibit A2 for the prefix code subset and Exhibit A3 for the suffix subset. Taken together, these codes use 3.79 binary digits, on the average, to encode each character.

The procedure for decompressing RAI code is interesting because it is surprisingly fast and efficient. Logically, one would think of scanning the bit stream to determine the end of a character and then to translate it into the normal IBM machine code. Because of the unique design of the RAI code, however, these steps can be combined.

From Exhibit A1 another table (IV) can be constructed by using the first four bits of each code as a number and multiplying by four to form an address (see Table IV opposite).

This table is imbedded in the decompression program in a slightly different form which permits direct translation. In the case of a 4 bit code, the address in bytes allows a direct jump to the table location which contains an instruction that places the correct character to be added to the output (decompressed) character stream. In those cases where the code is more than 4 bits, the instruction placed in the table transfers control to an instruction which looks at additional bits before translating. Note that

TABLE IV.

| Address | Character or Character Pair | Comment |
|---|---|---|
| 0 | ↗ | Four bit codes can be translated directly. |
| 4 | ,b | |
| 8 | E | |
| 12 | S | |
| 16 | A | |
| 20 | b | |
| 24 | I | |
| 28 | # | |
| 32 | M | |
| 36 | L | Five bit codes require examination of the next bit in the stream. |
| 36 | C | |
| 40 | O | |
| 40 | T | |
| 44 | ER | |
| 44 | R | |
| 48 | H | |
| 48 | D | |
| 52 | .# | |
| 52 | B | |
| 56 | P | Six bit codes require examination of the next two bits in the stream. |
| 56 | G | |
| 56 | AR | |
| 56 | W | |
| 60 | IN | Etc. |
| 60 | U | |
| 60 | AN | |
| Etc. | Etc. | |

from the first four bits of the code, or table address, the code length is determined, thus avoiding iterations. Note also that since the characters are ordered by frequency of occurrence, and that characters "↗" through "M" account for almost 40% of the characters to be translated, 40% of the time the translation will be accomplished with one step.

Other Statistical Coding Techniques — Two other techniques were used which take advantage of the statistical characteristic of the characters to be coded. Rather than using conventional 8 bit bytes or packed formats for numbers, binary codes were used where fixed length fields were desired. Greater compression resulted as compared with the RAI code technique. A sequence code, for example, was reduced from 2 bytes to 7 binary digits using this technique.

A Table-lookup technique was also used where the range of a field was much greater than the number of values it could contain. For example, certain names were in the uncompressed file which required 30 characters, yet only a few actual names existed (repeated many times). Certain types of codes also fell into this category and are recorded in the compressed file in shorthand notation. In one case, only 6 binary digits were required to store what originally required 2 bytes.

### Compression Module Integration

Assembly language routines were written for each of the five techniques described above which intercept and compress data from application programs to be stored on disk. Additional routines intercept read requests and expand it into the format expected by the application program. These routines were linked to the application programs as subroutines and although this necessitated recompilation, no significant programming changes were necessary.

### Compressed Record Formats

The increased use of variable length fields, partly caused by variable length binary codes, meant that selection of an optimum fixed record length was difficult. A statistical analysis was made of the cumulative effect of the various contributions to variability with the result that a base record length of 60 bytes (as compared with the original 280 bytes) was optimum and that, to a very high degree of confidence, only one trailer record would ever be required to accommodate overflow.

For simplicity, the compressed fields are placed in the new record in groups according to the compression technique employed. Since the variable and fixed length fields are segregated, there is little problem in calculating field boundaries so that the overload implied by decompressing variable length fields is small.

Record keys are similarly segregated and begin at a fixed place in the record. This is useful because most references to the file only require that the record key be known, since the most frequent task of the system is to search for matches on keys. To avoid unnecessary decompression, then, the search arguments themselves are compressed, and the search (actually performed by the ISAM modules) compares compressed data. Decompression is only performed after a "hit" has been found.

Great care was taken in the design and selection of the compression techniques applied to the record keys. Most significant was the concern for collating sequence and fixed length compression. Since ISAM requires that record keys adhere to fixed byte boundaries, there is a need to utilize techniques that will result in a fixed length compression of the record keys. The requirement to increase the compressed keys to the next byte boundary usually results in some latitude in the selection of techniques without penalty of added bytes and has made it possible to select techniques that will maintain the collating sequence of the original data. By maintaining the original file collating sequence, no special processing for conversion and file reorganization was required.

## IV. Results

As mentioned above, the basic record was compressed from 280 bytes to 60 bytes, a ratio of 4.7 to 1. Or saying it another way, the record was compressed to 21% of its original volume with no loss of information. Since trailer records will be about as frequent in the compressed format as they were in the uncompressed, the total volume of the Master File has been compressed from 250 million bytes to 54 million bytes. This means that the Master File, which originally was physically stored on a Data Cell, can now be held in less than two disk drives of an IBM/2314.

This compression was obtained for two main reasons. First, the "empty space" or blanks in the original records, primarily from unfilled fixed length fields containing variable length information, has been eliminated. Or more accurately, it has been eliminated from the individual fields and accumulated at the end of each record. This agglutination process results in less "empty" space because it is effectively shared by several fields, with longer than average fields consuming the space unused by shorter than average ones.

Second, the information stored in a record requires less space because more compact coding has been used. Overall, an average of 4.03 binary digits are required to record a character in name and address fields in the compressed record, as compared with almost 8* in the original Master File. Codes, amounts and other special fields required even fewer binary digits per character.

* Less than 8 binary digits were required because some fields used a packed format.

In addition to the elimination of empty space within fields, some fields were eliminated in their entirety. It was found that certain fields contained redundant information which existed in other fields. Often, the redundant information was generated within the system prior to storing the data rather than at output time. The data elimination in these fields is technically trivial, but produces small but significant results since reduction to zero represents infinite compression and major storage savings. Data compression studies usually act as a catalyst and result in constructive user attitudes and significant reductions in record content by elimination of unneeded information identified and volunteered by the users themselves.

A summary of the compression achieved is shown in Table V. The three general categories mentioned above (empty space, reduced representation, and elimination) are shown with the subdivision into the various techniques comprising reduced representation. The reader may note that it was not possible to eliminate all empty space since the DOS-ISAM environment requires fixed length records.

TABLE V. SUMMARY OF COMPRESSION

| Technique | Original Storage In Bytes | | Compressed Storage In Bytes | | Net Savings In Bytes | | Savings as % of Original Record | |
|---|---|---|---|---|---|---|---|---|
| Eliminated Empty Space | | 123.6 | | 6.6 | | 117.0 | | 41.8 |
| Reduced Representation | | | | | | | | |
| RAI Coding | 67.4 | | 34.0 | | 33.4 | | 11.9 | |
| Decimal to Binary | 25.0 | | 8.5 | | 16.5 | | 5.9 | |
| Table Look-up | 14.0 | | 5.9 | | 8.1 | | 2.9 | |
| Differencing | 24.0 | | 5.0 | | 19.0 | | 6.8 | |
| Sub Total | | 130.4 | | 53.4 | | 77.0 | | 27.5 |
| Field Elimination | | 26.0 | | 0.0 | | 26.0 | | 9.3 |
| Total Compression | | 280.0 | | 60.0 | | 220.0 | | 78.6 |

Operationally, programs which originally required about one hour to run, require only about 10 minutes with the compressed file. Of course, part of this improvement comes from use of a faster storage device made economically possible by compression. If the device component of the performance increase is removed, it is estimated that the performance of the system would be about the same as with the original file. That is, the compression and decompression CPU overhead is offset by improved channel and buffer performance.

The assembly language compression routines were written carefully to avoid waste space since limited core was available. As a result, the total area required for the five techniques, tables, and working storage were approximately 5k bytes.

In conclusion, this compression effort is considered successful from technical and economic viewpoints and demonstrates the feasibility of retroactively applying data compression techniques in a commercial environment.

## REFERENCES

(1) DE MAINE, P.A.D.: SPRINGER, G.K. The COPAK Compressor. In: File Organisation. Selected papers from File 68 -- An IAG Conference, Swets and Zeitlinger N.V., Amsterdam, 1969, 149-158.

(2) GAINES, HELEN FOUCHÉ. Cryptanalysis, Dover, New York, 1956.

(3) KAHN, DAVID. The Codebreakers, MacMillan Company, New York, 1967.

(4) KALLAB, J. A linear geographical code for management information systems. In: Computers and Automation. 17, 4 (April 68) 24-30.

(5) KENYON, W.S. Errors in Transmission of Compressed Data, PhD Thesis, Princeton Univ., Princeton, New Jersey, 1968.

(6) KERPELMAN, C. (Ed.) Proposed American National Standard; Identification of States .. ...for Information Interchange. In: Communications of the ACM, Vol. 13, No.8 (Aug.70), 514-515.

(7) KUKENHEIM, L. Coding of grammatical data. In: Sprachkunde und Informationsverarbeitung. No.1, 1963, 45-47 (German).

(8) LOHSE, E. (Ed.). Data code for calendar date for machine-to-machine data interchange. In: Communications ACM. 11, 4 (April 68) 273-274.

(9) MARRON, B.A.: DE MAINE, P.A.D. Automatic data compression. In: Communications of the A.C.M. 10, (November 1967), 711-715.

(10) MAURER, W.D. File compression using Huffman coding. In: Computing Methods in Optimization Problems, Vol.2, 247-256. Second International Conference on Computing Methods in Optimization Problems, San Ramo, Italy, September 1968, Academic Press, New York 1969.

(11) NEUMANN, P.G. Efficient Error-limiting Variable-length Codes. Thesis, Harvard University, Cambridge, Mass., 1961.

(12) NUGENT, W.R. Compression word coding techniques for information retrieval. In: Journal Library Automation 1, 4 (Dec.68), 250-260.

(13) NUGENT, W.R.: VEGH, A. Automatic word coding techniques for computer language processing. In: RADC-TDR-62-13, Vols. 1, 2.

(14) OVERHAGE, CARL F.J. (Ed.) Project INTREX Semiannual Activity Report PR-9, March 15, 1970, M.I.T., Page 28.

(15) PRATT, FLETCHER. Secret and Urgent, The Story of Codes and Ciphers, Blue Ribbon Books, Garden City, New York, 1939.

(16) SCHWARTZ, E.S. A Dictionary for Minimal Redundancy Encoding, JACM 10, (1963) 413-439.

(17) SCHWARTZ, E.S.: KLEIBOEMER, A.J. A language element for compression coding, In: Information and Control, 10, 3 (March 67) 315-333.

(18) SNYDERMAN, MARTIN: HUNT, BERNARD. The myriad virtues of text compaction. In: Datamation, 16, 16(1 December 70), p. 36.

(19) VERHOEFF, J. Error detecting and correcting codes for the decimal number system. In: Proceedings International Symposium on Automation of Population Register Systems, Vol. 1, 447-454.

(20) WRIGHT, M.A. Mechanizing a large index; appendix: the soundex code. In: The Computer Journal, 3(July 1960), p. 83.

EXHIBIT A1

SOURCE ALPHABET AND CODES
USED FOR NAME FIELDS

| Character | Frequency | Probability | Code | Character | Frequency | Probability | Code |
|---|---|---|---|---|---|---|---|
| ↗ | | .08726 | 0000 | J | 4,669 | .01344 | 1111110 |
| ,b | 21,654 | .06231 | 0001 | ON | 4,204 | .01210 | 1111111 |
| E | 21,572 | .06208 | 0010 | IL | 4,005 | .01152 | 0000 |
| S | 17,718 | .05099 | 0011 | K | 3,927 | .01130 | 0001 |
| A | 17,381 | .05002 | 0100 | F | 3,412 | .00982 | 0010 |
| b | 16,914 | .04867 | 0101 | EL | 3,372 | .00971 | 0011 |
| I | 14,523 | .04180 | 0110 | OR | 3,227 | .00928 | 0100 |
| # | 14,309 | .04117 | 0111 | EN | 3,084 | .00888 | 0101 |
| M | 13,990 | .04026 | 1000 | V | 2,666 | .00767 | 0110 |
| L | 13,779 | .03966 | 10010 | - | 2,381 | .00685 | 0111 |
| C | 12,078 | .03475 | 10011 | Z | 2,002 | .00576 | 1000 |
| O | 11,856 | .03412 | 10100 | b̸ | 1,000 | .00288 | 10010 |
| T | 11,853 | .03411 | 10101 | X | 499 | .00143 | 10011 |
| ER | 10,967 | .03156 | 10110 | 7 | 123 | .00035 | 10100 |
| R | 10,860 | .03125 | 10111 | Q | 117 | .00033 | 10101 |
| H | 10,747 | .03093 | 11000 | Ø | 102 | .00029 | 10110 |
| D | 10,318 | .02969 | 11001 | 1 | 97 | .00028 | 10111 |
| .# | 9,701 | .02792 | 11010 | 9 | 51 | .00015 | 11000 |
| B | 7,532 | .02167 | 11011 | 6 | 24 | .00007 | 11001 |
| P | 7,319 | .02107 | 111000 | ' | 221 | .00064 | 11010 |
| G | 7,191 | .02069 | 111001 | 3 | 19 | .00005 | 11011 |
| AR | 7,014 | .02018 | 111010 | 5 | 16 | --- | 111000 |
| W | 6,122 | .01762 | 111011 | 4 | 15 | | 111001 |
| IN | 5,983 | .01721 | 1111000 | 2 | 15 | | 111010 |
| U | 5,768 | .01660 | 1111001 | 8 | 6 | | 111011 |
| AN | 5,626 | .01619 | 1111010 | & | 1 | | 1111000 |
| Y | 5,414 | .01558 | 1111011 | • | - | | 1111001 |
| N | 5,203 | .01498 | 1111100 | Totals | 347,497 | .99969 | |
| AL | 4,850 | .01395 | 1111101 | | | | |

EXHIBIT A2

PREFIX CODES
USED IN ADDRESS FIELDS

| Character | Frequency | Probability | Code |
|---|---|---|---|
| b̲ | 29,500 | .1965 | 000 |
| ↗ | 24,010 | .1600 | 001 |
| 1 | 14,693 | .0980 | 010 |
| 2 | 12,945 | .0863 | 0110 |
| 3 | 9,196 | .0612 | 0111 |
| • | 9,000 | .0600 | 1000 |
| 5 | 7,907 | .0527 | 1001 |
| 0 | 7,639 | .0508 | 1010 |
| 4 | 7,319 | .0487 | 1011 |
| 6 | 5,858 | .0397 | 11000 |
| 7 | 4,607 | .0307 | 11001 |
| 8 | 4,229 | .0282 | 11010 |
| N | 4,000 | .0267 | 11011 |
| 9 | 3,534 | .0235 | 11100 |
| W | 1,500 | .0100 | 11101 |
| S | 1,500 | .0100 | 11110 |
| E | 1,500 | .0100 | 11111 |
| Totals | 148,937 | .9930 | |

EXHIBIT A3
SUFFIX CODES -- USED FOR ADDRESS FIELDS

| Character | Frequency | Probability | Code |
|---|---|---|---|
| E | 18,824 | .0858 | 0000 |
| R | 15,931 | .0726 | 0001 |
| T | 15,866 | .0723 | 0010 |
| N | 14,489 | .0660 | 0011 |
| A | 13,633 | .0622 | 0100 |
| b̲ST.# | 12,500 | .0570 | 0101 |
| b̲ | 12,220 | .0557 | 0110 |
| O | 11,898 | .0542 | 0111 |
| S | 11,196 | .0510 | 1000 |
| L | 9,875 | .0449 | 10010 |
| # | 8,510 | .0387 | 10011 |
| D | 8,416 | .0383 | 10100 |
| ↗ | | .0360 | 10101 |
| I | 7,748 | .0352 | 10110 |
| H | 7,397 | .0337 | 10111 |
| C | 5,021 | .0228 | 11000 |
| M | 4,256 | .0194 | 11001 |
| W | 3,727 | .0170 | 11010 |
| G | 3,509 | .0160 | 11011 |
| U | 3,449 | .0157 | 111000 |
| P | 3,278 | .0149 | 111001 |
| B | 3,149 | .0143 | 111010 |
| b̲AVE.# | 3,000 | .0137 | 111011 |
| K | 2,601 | .0118 | 1111000 |
| • | 2,430 | .0111 | 1111001 |
| Y | 2,354 | .0107 | 1111010 |
| F | 2,235 | .0102 | 1111011 |
| V | 1,702 | .0078 | 1111100 |
| ' | 974 | .0044 | 1111101 |
| 0 | 800 | .0036 | 1111110 |
| 1 | 800 | .0036 | 1111111 |
| 2 | 800 | .0037 | 0000 |
| 3 | 800 | .0036 | 0001 |
| 4 | 800 | .0036 | 0010 |
| 5 | 800 | .0037 | 0011 |
| 6 | 800 | .0036 | 0100 |
| 7 | 800 | .0036 | 0101 |
| 8 | 800 | .0037 | 0110 |
| 9 | 800 | .0036 | 0111 |
| J | 441 | .0020 | 1000 |
| Z | 410 | .0019 | 10010 |
| - | 257 | .0012 | 10011 |
| X | 249 | .0011 | 10100 |
| Q | 121 | .0005 | 10101 |
| @ | 42 | .0002 | 10110 |
| & | 00 | .0000 | 10111 |
| Totals | 219,708 | 1.0006 | |