G. Schay
N. Raver

A Method for Key-to-Address Transformation

Abstract: Techniques used in the theory of error-correcting codes are applied to solve the problem of addressing a large file. This novel approach to the file addressing problem is illustrated with a specific design to show feasibility. Its effectiveness is further illustrated by comparing test results obtained from a simulated calculation, which used typical data, against values calculated from an ideal model.

Introduction

The problem of addressing a large file is often handled by using a portion of each record in the file for unique identification. This portion is usually called the *key*, and could be a number, as in shop-order numbers, or a group of alphanumerical characters, as in records identified by name and address. This paper develops a method for attacking this problem using the tools developed in the theory of error-correcting codes.

In the case where the keys consist of a sequential set of numbers, the record key can readily be converted to a memory address if all records are of a fixed length. More generally, however, the record keys are not sequential, and only a small fraction of all allowable keys are employed; within this fraction, there is no orderly selection. Examples of this type of key are parts numbers, tool numbers, and word orders. For example, if the record key were to consist of 20 alphanumeric characters, then there would be $2^{120} \cong 10^{36}$ possible keys, assuming 6 bits per character. For this more general case, usually no simple relation is definable between record key and an assignable storage address.

Addressing a file made up of such records can be handled by two general methods; the first commonly known approach is the use of tables to store the relationship between record key and memory address. Whenever a record has to be accessed in the file, its key is used to scan the tables to find a starting address.[1] This approach is often called the *table-lookup method.*

The second method often employed is to transform the record key into a storage address by techniques which effectively compress the allowable range of record keys into the allowable range of storage addresses. These methods are often called *key-to-address transformations* or randomizing schemes.[2-7] For convenience, we can divide these transformations into two classes: those which require prior knowledge of the key set and those which require no prior examination of the key set. Examples of the first class are given in Refs. 2 and 4. The "ideal" result we might expect for this class would be a uniform assignment of records to all storage locations, since the complete key set is available for inspection prior to the formulation of our transformation. For example, assume that a file consists of 10^5 records, each being 100 characters in length, and also that there are 10^5 storage locations, each 100 characters in length. A uniform transformation would ideally assign all 10^5 keys uniformly to all 10^5 locations. Clearly any records added or deleted to an existing file require changes to the transformation if the ideal is to be maintained. To our knowledge, no such scheme has yet achieved the "uniform" ideal.

The second class does not assume *a priori* knowledge of the key set and employs a fixed transformation which uniquely derives a storage address from each key. The "ideal" model for this class is more imperfect and assumes that the transformed keys will be *randomly* assigned to storage locations, hence the term "randomizing." For random transformations, the possibility of two or more keys being transformed to the same address must, of course, exist and such keys are called *synonyms.* Examples of this class have been given previously.[3,4,6]

The method of transformation used in this paper employs the tools that have been developed in the theory of error-correcting codes. The presentation in this paper has been oriented primarily to those concerned with the file problem and most of the material will be familiar to the specialist in error-correcting codes. The first part of the paper develops the equivalence between a certain formulation of the file problem and the main problem of the theory of error-correcting codes; the theory of the actual transformation is then derived. Following this, the remainder of the paper gives an illustrative equipment

design to indicate the economy of the technique plus a derivation of performance for an "ideal" or random transformation.

A processor with a large file could be equipped with this design so that hardware accomplishes the transformation rapidly under program control by adding an extra instruction TRANSFORM to the central machine. Alternatively, the technique can of course by accomplished using the existing order code of the machine in a fixed subroutine.

The problem

Addressing a random access memory by means of key-to-address transformation requires an algorithm for transforming or mapping the key efficiently onto memory storage locations. We have chosen for the ideal result the case where the transformed keys have a "uniform" probability distribution over all buckets. This ideal model is rigorously defined in the section on test results.

We define the set of all possible keys as the set K, which is generally very large, as illustrated in the Introduction. Generally, the keys are strings of length n of symbols, each symbol being taken from an alphabet of 2^q symbols, where q is the number of bits per character. The actual keys appearing in a particular file form a small subset S of the set K. We further assume that the memory has M addresses numbered from 1 to M and will call this set A. Obviously, the number of elements of any S must not exceed M.

An examination of actual key sets S reveals that a typical characteristic is the occurrence of *clusters*. For example, ABCD and ACBE are at Distance 3 from each other in any alphabet which contains A, B, C, D, and E. This distance definition, however, depends on the representation. For example, if we code A, B, C, D, E respectively as 000, 001, 010, 011, 100, then ABCD becomes 000001010011 and ACBE becomes 000010001100, and the distance of the two keys in this two-symbol representation is 7 and not 3. For this reason we use a fixed representation, that is, a fixed q. Clusters can be defined now as sets of keys, which are near to each other. To be more precise, we define a cluster of diameter d as any set of keys, in which the maximum distance between pairs is d. For example, the following set of names is a cluster of Diameter 4 in any alphabet containing A, B, C, N, O, R, U, W:

BR AU N A B

BR AU N A C

BR AU N B C

BR OW N A A

BR OW N B A

Following Lin,[3] we also assume that the transform must destroy the clusters in the set S and disperse the elements of a cluster amongst the storage buckets. Our transformation must map the elements of any cluster whose diameter is less than d into a different address and d should be as large as possible. This maximal d will be denoted by D. We have, therefore, assumed that *the only restriction in any input set S is that no cluster will exceed diameter D.*

This outlined formulation makes the problem equivalent to the main problem of the theory of error-correcting codes. The method of Varsharmov and Gilbert can be adapted to find D, while the Bose-Chaudhuri method is used to construct an actual transformation.[8]

The basic formulation

Our aim is to partition the given set K into M subsets in such a way that the elements in each subset are at least distance D away from each other. This is equivalent to our original formulation, since numbering these subsets from 1 to M and considering the transformation which maps every key into the number of the subset containing it, we have a transformation which maps any two keys, that are at a distance less than D from each other, into different numbers.

To be able to apply the familiar methods of algebra, K and A will be regarded as respectively n and m dimensional vector spaces over the field of 2^q elements. Since we take the symbols of the alphabet as the elements of a field, one of them is the 0 of the field. The weight $w(x)$ of any key vector of x is then defined as the number of its nonzero components. The distance $d(x, y)$ of any two keys x and y equals the weight of $x - y$.

We seek a linear transformation from a key $(a_1, \cdots, a_n)$ to an address $(p_1, \cdots, p_m)$, i.e., one of the form

$$\sum_{j=1}^{n} t_{ij}a_j = p_i \qquad \text{for } i = 1, \cdots, m. \tag{1}$$

Here (t_{ij}) is the required transformation matrix and it has to have m rows and n columns.

Theorem

The distance between every pair of keys mapped into the same address by the transformation matrix (t_{ij}) is larger than D if, and only if, every D columns of the matrix are independent. This implies in particular that D has to be less than or equal to m.

Proof. We want distances only larger than D to exist between keys mapped into the same address. Thus the weight of the difference of two such keys has to be larger than D. Let us consider then two keys a and b, which map into the same address p, and whose difference has weight D, that is, $n - D$ zero digits. The transformation formula (1) gives

$$\sum_{j=1}^{n} t_{ij}a_j - \sum_{j=1}^{n} t_{ij}b_j = \sum_{j=1}^{n} t_{ij}(a_j - b_j) = 0 \qquad \text{for } i = 1, \cdots, m, \tag{2}$$

since a and b map into the same p. Now if $n - D$ of the $(a_j - b_j)$'s are zero, then to satisfy our requirement that this should make all $(a_j - b_j)$'s zeros, the equations (2) should give at least D *independent* equations for the D $(a_j - b_j)$'s which were not assumed to be 0. This require-

ment is equivalent to saying that every D columns of the matrix t_{ij} have to be independent of each other.

For the particular case in which the alphabet consists of 2^6 symbols, that is, all operations are performed in the Galois field of 2^6 elements, and the number of elements of K is 2^{180} and that of A is 2^{30}, that is, $n = 180/6 = 30$ and $m = 30/6 = 5$. Varsharmov and Gilbert's method for the construction of such matrices gives $D = 5$. (See Ref. 8).

The above theorem establishes the relationship between the coding and addressing problems, since this class of matrices, deduced here for the latter, plays a basic role in coding theory.

One such transformation can be constructed by the method of Bose and Chaudhuri[8,9] using a polynomial representation equivalent to the matrix representation given here. This method leads to a simple implementation. The keys and addresses have to be represented in it as polynomials:

$$K(x) = \sum_{i=1}^{n} a_i x^{i-1}, \qquad R(x) = \sum_{i=1}^{n} p_i x^{i-1}. \tag{3}$$

Let α be a primitive element of $GF(2^q)$ (that is $\alpha^{2^q-1} = 1$ but $\alpha^i \neq 1$ for $i < 2^q - 1$) and consider the polynomial

$$g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{d-1}) = \sum_{i=0}^{d-1} g_i x^i, \tag{4}$$

where $d \leq D$.

The Bose-Chaudhuri theorem states that if $R(x)$ is the remainder of the division of $K(x)$ by $g(x)$, that is,

$$K(x) = Q(x)g(x) + R(x), \text{ degree of } R(x) < d - 1, \tag{5}$$

then the minimum distance between two keys giving the same R is at least d.

Implementation

The basic equation to be implemented is Eq. (5). It may be written as

$$\frac{K(x)}{g(x)} = Q(x) + \frac{R(x)}{g(x)}. \tag{6}$$

The division of two polynomials can be carried out by the equipment shown in Fig. 1. Peterson gives additional background on hardware implementation.[8] The K register consists of K_0, $K_1 \cdots K_5$, and each K unit contains 6 flip-flops. The individual coefficients of $K(x)$ are successively shifted into the K register so that after 5 such shifts, each stage of the K register will contain:

K_0 K_1 K_2 K_3 K_4

a_{26} a_{27} a_{28} a_{29} a_{30}

The output of the K_4 stage is coupled through the S unit to the various G units. Each G_i unit multiplies the coefficient in K_4 by the coefficient g_i of $G(x)$. No G_5 unit is needed since $g_5 = 1$.

Each G unit consists of a logic array which performs the straight Boolean multiplication directly. Table 1 shows the direct multiplication of two elements of a $GF(2^6)$ for the general case. Table 2 gives the specific Boolean equations for G_0 and G_4. The worst case is G_4, which requires 6 successive additions for d_5, or a propagation time for 6 stages of EXCLUSIVE OR's. Hence, it is reasonable to assume that this method can accomplish the multiplication in one clock time t_c. Total subcycle time for one K register shift followed by the multiplication and addition should not exceed two clock times.

The S unit in Fig. 1 merely regenerates the K_4 outputs with sufficient power to drive G_0 through G_4, and all the

Figure 1 **Gross logic for calculating $R(x)$.**

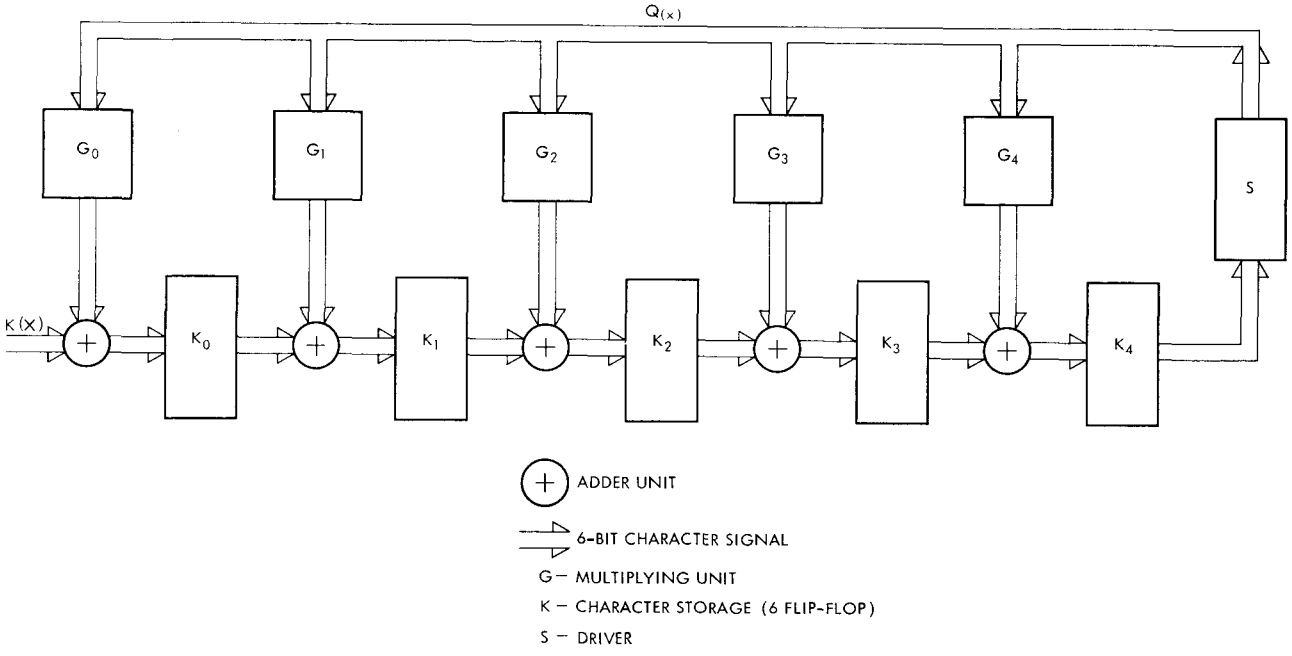


Table 1 **Multiplication of two elements in $GF(2^6)$.**

$$\alpha^{n+m} = \alpha^n\alpha^m = (b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3 + b_4\alpha^4 + b_5\alpha^5)(c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3 + c_4\alpha^4 + c_5\alpha^5),$$

using $1 + \alpha = \alpha^6, \alpha + \alpha^2 = \alpha^7, \cdots \alpha^4 + \alpha^5 = \alpha^{10}$

$$\alpha^{n+m} = d_0 + d_1\alpha^1 + d_2\alpha^2 + d_3\alpha^3 + d_4\alpha^4 + d_5\alpha^5.$$

Expanding, reducing and collecting like terms then gives:

$$d_0 = b_0c_0 + b_5c_1 + b_4c_2 + b_3c_3 + b_2c_4 + b_1c_5$$

$$d_1 = b_1c_0 + b_0c_1 + b_5c_1 + b_4c_2 + b_3c_3 + b_2c_4 + b_1c_5 + b_5c_2 + b_4c_3 + b_3c_4 + b_2c_5$$

$$d_2 = b_2c_0 + b_1c_1 + b_0c_2 + b_5c_2 + b_4c_3 + b_3c_4 + b_2c_5 + b_5c_3 + b_4c_4 + b_3c_5$$

$$d_3 = b_3c_0 + b_2c_1 + b_1c_2 + b_0c_3 + b_5c_3 + b_4c_4 + b_3c_5 + b_5c_4 + b_4c_5$$

$$d_4 = b_4c_0 + b_3c_1 + b_2c_2 + b_1c_3 + b_0c_4 + b_5c_4 + b_4c_5 + b_5c_5$$

$$d_5 = b_5c_0 + b_4c_1 + b_3c_2 + b_2c_3 + b_1c_4 + b_0c_5 + b_5c_5.$$

G units are networks of Boolean logic consisting of approximately 53 EXCLUSIVE OR gates.

The equipment in Fig. 1 requires the following:

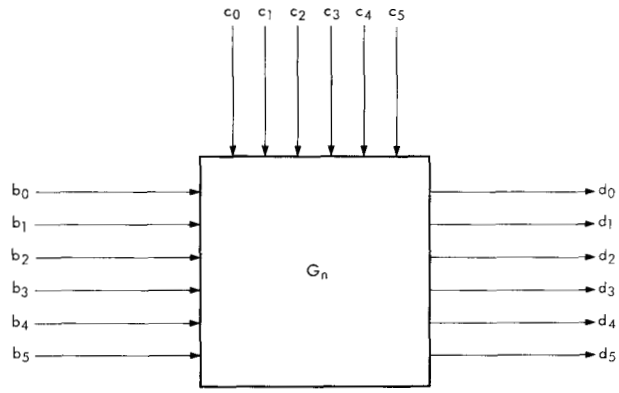
	Number of flip-flops	*Number of* EXCLUSIVE OR *gates*
K register	30	0
G unit	0	53
Adder	0	30
	30	83

Some test results

In order to illustrate the effectiveness of the technique, a test run was made using raw data taken from a typical customer. The transformation was simulated on an IBM 7090 for selected ranges of parameters. These results are best evaluated after the ideal model is developed.

A bucket is defined as the smallest uniquely addressable segment of memory and possible examples are a track (2800 characters) or a cylinder (40 tracks × 2800 char./track = 112,000 char.) in the IBM 1301. A *cell* is defined as a segment of a bucket; as an example, if each cell is assigned 100 characters, then for the IBM 1301, each track will have 2800/100 = 28 cells and each cylinder will have 112,000/100 = 1120 cells. Records are assigned to storage by transforming the record key and using the transformed result to designate a bucket. Within the assigned bucket, the record will occupy a cell or group of cells. Subsequent accesses to this record are accomplished by transformation of the record key and then searching through the designated bucket for the cells which contain the record. When synonyms occur, they are assigned to the same bucket and when bucket capacity is exceeded, remaining synonyms are considered overflow.[10]

Table 2 **Boolean functions for G multipliers.**



Note: for all five G units, need about 53 EXCLUSIVE OR's.

$$g(x) = g_0 + g_1x + g_2x^2 + g_3x^3 + g_4x^4 + g_5x^5$$
$$= \alpha^{15} + \alpha^3x + \alpha^{11}x^2 + \alpha^8x^3 + \alpha^{57}x^4 + x^5$$

Set $b_0, b_1, \cdots b_5$, to g_n values for each n.

$G_0, g_0 = \alpha^{15}, \quad b_3 = b_5 = 1, \quad b_0 = b_1 = b_2 = b_4 = 0.$

$$d_0 = c_3 + c_1$$
$$d_1 = c_1 + c_3 + c_2 + c_4$$
$$d_2 = c_2 + c_4 + c_3 + c_5$$
$$d_3 = c_0 + c_3 + c_5 + c_4$$
$$d_4 = c_1 + c_4 + c_5$$
$$d_5 = c_0 + c_2 + c_5.$$

$G_4, g_4 = \alpha^{57} \quad b_0 = 0, \quad b_1 = b_2 = b_3 = b_4 = b_5.$

$$d_0 = c_1 + c_2 + c_3 + c_4 + c_5$$
$$d_1 = c_0 + c_1$$
$$d_2 = c_0 + c_1 + c_2$$
$$d_3 = c_0 + c_1 + c_2 + c_3$$
$$d_4 = c_0 + c_1 + c_2 + c_3 + c_4$$
$$d_5 = c_0 + c_1 + c_2 + c_3 + c_4 + c_5.$$

Assume the file contains r independent records and that each record is a call in length. Also, assume that the number of storage locations or cells in memory is N. If the transformation employed gave ideal results, i.e., the transformed address set were random, then it can be shown that the distribution of the number of records, k, having an arbitrary common address would be approximately

$$p_k = \frac{\left(\frac{r}{N}\right)^k \exp\left[-\frac{r}{N}\right]}{k!}, \qquad k = 0, 1, 2, \cdots \quad (7)$$

Here p_k is the probability of finding k records with the same calculated address or k synonyms.

If the N storage locations or cells are grouped into buckets, then if b cells are assigned per bucket, there will be N/b buckets. Since we have assumed constant record length of one cell, then storage density or effective utilization of storage space will be $d = r/N$. It can be shown that the average number of overflow records or excess synonyms that will be assigned to a bucket of b cells will be given by:

$$T = \frac{1}{b}\left\{b(d-1) + \sum_{k=0}^{k=b-1} (b-k)\,\frac{(bd)^k \exp\,[-bd]}{k!}\right\}. \quad (8)$$

Here T is given normalized to the number of storage locations N, hence $T \times 100\%$ will give percent overflow. For example, if $T = 0.1$ and $N = 1000$, then 10%, or 100 records, will overflow their assigned or home buckets. This formula is derived in Appendix I.

In our test runs the input data involved numbers such as the following:

Cylinder Assy.	1025AA-71-C-S1
Cylinder Assy.	1026AA-72-B-S1

The test runs showed the results to correlate quite well with the random model described by Eqs. 6 and 7. Table 3 gives a typical tabulation of the numbers of buckets versus k assigned records; this comparison is for the case of $d = 1$, $N = 4096$.

Table 3 **Sample of test results against random (Poisson distribution).**

Number of records to a bucket k	*Measured number of buckets having k records*	*Number of buckets for random set, from Eq. (7)*
0	1501	1510
1	1513	1510
2	750	755
3	257	252
4	63	63
5	12	13
6	0	3

In the tests we also selected different bucket sizes to determine the number of records that would overflow their home bucket. Figure 2 depicts some test results from the raw data; the values calculated from Eq. (8) are superimposed.

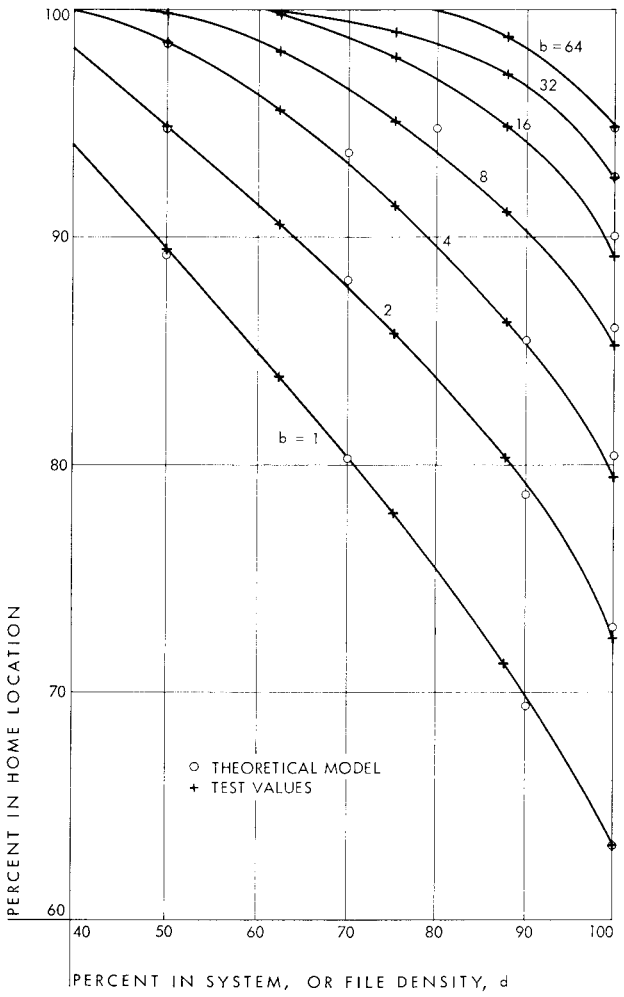


Figure 2 **Test runs giving overflow vs file density.**

Although the present study did not examine analytically how random the transformed set will be, this topic probably deserves further attention. It is, of course, the characteristics of the original key set which determine the randomness of the transformed results.

Conclusions

The present paper demonstrates how the theory of error-correcting codes can be applied to the problem of addressing a file. The necessary hardware is shown to be readily implemented, and the results closely approximate the ideal model.

Acknowledgments

The authors wish to express their thanks to F. P. Palermo and M. Hanan for many interesting discussions, and in particular for their suggestion to use the Bose-Chaudhuri codes for the implementation. The raw data for the test

results were supplied by R. M. Simons. The programing effort to develop the simulated test results was accomplished by G. H. Bean.

Appendix: Derivation of the overflow

The probability that k records will be assigned to a bucket of b cells is given by

$$\frac{\left(\frac{br}{N}\right)^k \exp\left[-\frac{br}{N}\right]}{k!}.$$

This can be demonstrated by taking the b-fold convolution of Eq. (7), the Poisson distribution. From this, the expected overflow T_b of records in a bucket of b cells will be:

$$T_b = \sum_{k=b}^{k=\infty} (k - b) \frac{\left(\frac{br}{N}\right)^k \exp\left[-\frac{br}{N}\right]}{k!}.$$

Expanding and combining terms and noting that

$$\sum_{k=0}^{k=\infty} k \frac{\left(\frac{br}{N}\right)^k \exp\left[-\frac{br}{N}\right]}{k!} = b\frac{r}{N}, \quad \text{(Poisson mean) and}$$

$$\sum_{k=0}^{k=\infty} b \frac{\left(\frac{br}{N}\right)^k \exp\left[-\frac{br}{N}\right]}{k!} = b \quad \text{(sum of Poisson is 1),}$$

we obtain

$$T_b = \left\{\frac{br}{N} - b - \sum_{k=0}^{k=b-1} (k - 1) \frac{\left(\frac{br}{N}\right)^k \exp\left[-\frac{br}{N}\right]}{k!}\right\}.$$

Since we have N/b buckets and N cells or storage locations, total overflow normalized to the number of cells is $T = T_b \times (N/b) \times (1/N)$. Hence total overflow T is given by

$$T = \frac{1}{b}\left\{b\left(\frac{r}{N} - 1\right) + \sum_{k=0}^{k=b-1} (b - k) \frac{\left(\frac{br}{N}\right)^k \exp\left[-\frac{br}{N}\right]}{k!}\right\}.$$

Setting $d = r/N$ as a measure of storage density this becomes

$$T = \frac{1}{b}\left\{b(d - 1) + \sum_{k=0}^{k=b-1} (b - k) \frac{(bd)^k \exp\left[-bd\right]}{k!}\right\}.$$

References

1. W. W. Peterson, *IBM Journal* **1,** 130 (1957).
2. H. M. Sierra, unpublished report, February, 1960.
3. A. D. Lin, unpublished report, May, 1961.
4. *A General Approach to Automatic Programmed Address Conversion,* IBM Form No. J20-0235.
5. *IBM 1410 Data Processing System—IBM 1301 Disk Storage,* Reference Manual, IBM Form A22-6670.
6. C. A. Olsen, unpublished report, November, 1961.
7. S. Muroga, unpublished report, April, 1961.
8. W. W. Peterson, *Error Correcting Codes,* MIT Press and John Wiley & Sons, New York, 1961.
9. M. Hanan and F. P. Palermo, this journal, p. 127.
10. W. G. Dye, unpublished report, April, 1961.

Received March 20, 1962