Space/Time Trade-offs in Hash Coding with Allowable Errors

BURTON H. BLOOM
Computer Usage Company, Newton Upper Falls, Mass.

In this paper trade-offs among certain computational factors in hash coding are analyzed. The paradigm problem considered is that of testing a series of messages one-by-one for membership in a given set of messages. Two new hash-coding methods are examined and compared with a particular conventional hash-coding method. The computational factors considered are the size of the hash area (space), the time required to identify a message as a nonmember of the given set (reject time), and an allowable error frequency.

The new methods are intended to reduce the amount of space required to contain the hash-coded information from that associated with conventional methods. The reduction in space is accomplished by exploiting the possibility that a small fraction of errors of commission may be tolerable in some applications, in particular, applications in which a large amount of data is involved and a core resident hash area is consequently not feasible using conventional methods.

In such applications, it is envisaged that overall performance could be improved by using a smaller core resident hash area in conjunction with the new methods and, when necessary, by using some secondary and perhaps time-consuming test to "catch" the small fraction of errors associated with the new methods. An example is discussed which illustrates possible areas of application for the new methods.

Analysis of the paradigm problem demonstrates that allowing a small number of test messages to be falsely identified as members of the given set will permit a much smaller hash area to be used without increasing reject time.

KEY WORDS AND PHRASES: hash coding, hash addressing, scatter storage, searching, storage layout, retrieval trade-offs, retrieval efficiency, storage efficiency
CR CATEGORIES: 3.73, 3.74, 3.79

Introduction

In conventional hash coding, a hash area is organized into cells, and an iterative pseudorandom computational process is used to generate, from the given set of messages, hash addresses of empty cells into which the messages are then stored. Messages are tested by a similar process of iteratively generating hash addresses of cells. The contents of these cells are then compared with the test messages. A match indicates the test message is a member of the set; an empty cell indicates the opposite. The reader is assumed to be familiar with this and similar conventional hash-coding methods [1, 2, 3].

The new hash-coding methods to be introduced are suggested for applications in which the great majority of messages to be tested will not belong to the given set. For these applications, it is appropriate to consider as a unit of time (called *reject time*) the average time required to classify a test message as a nonmember of the given set. Furthermore, since the contents of a cell can, in general, be recognized as not matching a test message by examining only part of the message, an appropriate assumption will be introduced concerning the time required to access individual bits of the hash area.

In addition to the two computational factors, reject time and space (i.e. hash area size), this paper considers a third computational factor, allowable fraction of errors. It will be shown that allowing a small number of test messages to be falsely identified as members of the given set will permit a much smaller hash area to be used without increasing the reject time. In some practical applications, this reduction in hash area size may make the difference between maintaining the hash area in core, where it can be processed quickly, or having to put it on a slow access bulk storage device such as a disk.

Two methods of reducing hash area size by allowing errors will be introduced. The trade-off between hash area size and fraction of allowable errors will be analyzed for each method, as well as their resulting space/time trade-offs.

The reader should note that the new methods are not intended as alternatives to conventional hash-coding methods in any application area in which hash coding is currently used (e.g. symbol table management [1]). Rather, they are intended to make it possible to exploit the benefits of hash-coding techniques in certain areas in which conventional error-free methods suffer from a need for hash areas too large to be core resident and consequently found to be infeasible. In order to gain substantial reductions in hash area size, without introducing excessive reject times, the error-free performance associated with conventional methods is sacrificed. In application areas where error-free performance is a necessity, these new methods are not applicable.

A Sample Application

The type of application in which allowing errors is expected to permit a useful reduction in hash area size may be characterized as follows. Suppose a program must perform a computational process for a large number of distinct cases. Further, suppose that for a great majority of these cases the process is very simple, but for a small

Work on this paper was in part performed while the author was affiliated with Computer Corporation of America, Cambridge, Massachusetts.

difficult-to-identify minority the process is very complicated. Then, it might be useful to hash code identifiers for the minority set of cases so that each case to be processed could be more easily tested for membership in the minority set. If a particular case is rejected, as would happen most of the time, the simple process would be used. If a case is not rejected, it could then be subjected to a follow-up test to determine if it is actually a member of the minority set or an "allowable error." By allowing errors of this kind, the hash area may be made small enough for this procedure to be practical.

As one example of this type of application, consider a program for automatic hyphenation. Let us assume that a few simple rules could properly hyphenate 90 percent of all English words, but a dictionary lookup would be required for the other 10 percent. Suppose this dictionary is too big to fit in available core memory and is therefore kept on a disk. By allowing a few words to be falsely identified as belonging to the 10 percent, a hash area for the 10 percent might be made small enough to fit in core. When an "allowable error" occurred, the test word would not be found on the disk, and the simple rules could be used to hyphenate it. Needless disk access would be a rare occurrence with a frequency related to the size of the core resident hash area. This sample application will be analyzed in detail at the end of this paper.

A Conventional Hash-Coding Method

As a point of departure, we will review a conventional hash-coding method where no errors are permitted. Assume we are storing a set of n messages, each b bits long. First we organize the hash area into h cells of $b + 1$ bits each, $h > n$. The extra bit in each cell is used as a flag to indicate whether or not the cell is empty. For this purpose, the message is treated as a $b + 1$ bit object with the first bit always set to 1. The storing procedure is then as follows:

> Generate a pseudorandom number called a *hash address*, say k, $(0 \leqslant k \leqslant h - 1)$ in a manner which depends on the message under consideration. Then check the kth cell to see if it is empty. If so, store the message in the kth cell. If not, continue to generate additional hash addresses until an empty cell is found, into which the message is then stored.

The method of testing a new message for membership is similar to that of storing a message. A sequence of hash addresses is generated, using the same random number generation technique as above, until one of the following occurs.

1. A cell is found which has stored in it the identical message as that being tested. In this case, the new message belongs to the set and is said to be *accepted.*

2. An empty cell is found. In this case the new message does not belong to the set and is said to be *rejected.*

Two Hash-Coding Methods with Allowable Errors

Method 1 is derived in a natural way from the conventional error-free method. The hash area is organized into cells as before, but the cells are smaller, containing a code instead of the entire message. The code is generated from the message, and its size depends on the permissible fraction of errors. Intuitively, one can see that the cell size should increase as the allowable fraction of errors gets smaller. When the fraction of errors is sufficiently small (approximately 2^{-b}), the cells will be large enough to contain the entire message itself, thereby resulting in no errors. If P represents the allowable fraction of errors, it is assumed that $1 \gg P \gg 2^{-b}$.

Having decided on the size of cell, say $c < b$, chosen so that the expected fraction of errors will be close to and smaller than P, the hash area is organized into cells of c bits each. Then each message is coded into a c-bit code (not necessarily unique), and these codes are stored and tested in a manner similar to that used in the conventional error-free method. As before, the first bit of every code is set to 1. Since the codes are not unique, as were the original messages, errors of commission may arise.

Method 2 completely gets away from the conventional concept of organizing the hash area into cells. The hash area is considered as N individual addressable bits, with addresses 0 through $N - 1$. It is assumed that all bits in the hash area are first set to 0. Next, each message in the set to be stored is hash coded into a number of distinct bit addresses, say $a_1, a_2, \cdots, a_d$. Finally, all d bits addressed by a_1 through a_d are set to 1.

To test a new message a sequence of d bit addresses, say $a_1', a_2', \cdots, a_d'$, is generated in the same manner as for storing a message. If all d bits are 1, the new message is accepted. If any of these bits is zero, the message is rejected.

Intuitively, it can be seen that up to a point of diminishing returns, the larger d is, the smaller will be the expected fraction of errors. This point of diminishing returns occurs when increasing d by 1 causes the fraction of 1 bits in the hash field to grow too large. The increased a priori likelihood of each bit accessed being a 1 outweighs the effect of adding the additional bit to be tested when half the bits in the hash field are 1 and half are 0, as will be shown later in this paper. Consequently, for any given hash field size N, there is a minimum possible expected fraction of errors, and method 2 therefore precludes the error-free performance possible by modifying method 1 for a very small allowable fraction of errors

Computational Factors

Allowable Fraction of Errors. This factor will be analyzed with respect to how the size of the hash area can be reduced by permitting a few messages to be falsely identified as members of a given set of messages. We repre-

sent the fraction of errors by

$$P = (n_a - n)/(n_t - n), \tag{1}$$

where: n_a is the number of messages in the message space that would be accepted as members of the given set; n is the number of messages in the given set; and n_t is the total number of distinct messages in the message space.

Space. The basic space factor is the number of bits, N, in the hash area. By analyzing the effect on the time factor of changing the value of N, a suitable normalized measure of the space factor will be introduced later. Using this normalized measure will separate the effects on time due to the number of messages in the given message set and the allowable fraction of errors from the effects on time due to the size of the hash area. This separation of effects will permit a clearer picture of the space/time trade-offs to be presented.

Time. The time factor is the average time required to reject a message as a member of the given set. In measuring this factor, the unit used is the time required to calculate a single bit address in the hash area, to access the addressed bit, and to make an appropriate test of the bit's contents.

For the conventional hash-coding method, the test is a comparison of the addressed bit in the hash area with the corresponding bit of the message. For method 1, the test is a comparison of the hash area bit with a corresponding bit of a code derived from the message. For method 2, the test is simply to determine the contents of the hash area bit; e.g. is it 1? For the analysis to follow, it is assumed that the unit of time is the same for all three methods and for all bits in the hash area.[1]

The time factor measured in these units is called the normalized time measure, and the space/time trade-offs will be analyzed with respect to this factor. The normalized time measure is

$$T = \underset{m_i \in \hat{a}}{\text{mean}} (t_i), \tag{2}$$

where: M is the given set of messages; a is the set of messages identified (correctly or falsely) as members of M; $\bar{a}$ is the set of messages identified as nonmembers of M; m_i is the ith message; and t_i is the time required to reject the ith message.

[1] The reader should note that this is a very strong assumption, since the time to generate the successive hash-coded bit addresses used in method 2 is assumed to be the same as the time to successively increment the single hash-coded cell address used in method 1. Furthermore, conventional computers are able to access memory and make comparisons in multibit chunks, i.e. bytes or words. The size of a chunk varies from one machine to another. In order to establish the desired mathematical comparisons, some fixed unit of accessibility and comparability is required, and it is simplest to use a single bit as this unit. If the reader wishes to analyze the comparative performance of the two methods for a machine with a multibit unit and wishes to include the effects of multiple hash codings per message, this can readily be done by following a similar method of analysis to the one used here for a single-bit unit.

Analysis of the Conventional Hash-Coding Method

The hash area has N bits and is organized into h cells of $b + 1$ bits each, of which n cells are filled with the n messages in M. Let ϕ represent the fraction of cells which are empty. Then

$$\phi = \frac{h - n}{h} = \frac{N - n \cdot (b + 1)}{N}. \tag{3}$$

Solving for N yields

$$N = \frac{n \cdot (b + 1)}{1 - \phi}. \tag{4}$$

Let us now calculate the normalized time measure, T. T represents the expected number of bits to be tested during a typical rejection procedure. T also equals the expected number of bits to be tested after a nonempty cell has been accessed and abandoned. That is, if a hash-addressed cell contains a message other than the message to be tested, on the average this will be discovered after, say, E bits are tested. Then the procedure, in effect, starts over again.

Since ϕ represents the fraction of cells which are empty, then the probability of accessing a nonempty cell is $(1 - \phi)$, and the probability of accessing an empty cell is ϕ. If an nonempty cell is accessed, the expected number of bits to be tested is $E + T$, since E represents the expected number of bits to be tested in rejecting the nonempty accessed cell, and T represents the expected number of bits to be tested when the procedure is repeated. If an empty cell is accessed, then only one bit is tested to discover this fact. Therefore

$$T = (1 - \phi)(E + T) + \phi. \tag{5}$$

In order to calculate a value for E, we note that the conditional probability that the first x bits of a cell match those of a message to be tested, and the $(x + 1)$th bit does not match, given that the cell contains a message other than the message to be tested, is $(\frac{1}{2})^x$. (The reader should remember that the first bit of a message always matches the first bit of a nonempty cell, and consequently the exponent is x rather than $x + 1$, as would otherwise be the case.) Thus, for $b \gg 1$, the expected value of E is approximated by the following sum:

$$\sum_{x=1}^{\infty} (x + 1) \cdot (\tfrac{1}{2})^x = 3. \tag{6}$$

Therefore

$$T = (3/\phi) - 2, \tag{7}$$

$$N = n \cdot (b + 1) \cdot \frac{T + 2}{T - 1}. \tag{8}$$

Equation (8) represents the space/time trade-off for the conventional hash-coding method.

Analysis of Method 1

The hash area contains N' bits and is organized into cells of c bits each. In a manner analogous to the conventional method, we establish the following equations:

$$\phi' = \frac{N' - n \cdot c}{N'} = \text{the fraction of empty cells.} \quad (9)$$

$$N' = \frac{n \cdot c}{1 - \phi'}. \quad (10)$$

$$T' = (3/\phi') - 2. \quad (11)$$

$$N' = n \cdot c \cdot \frac{T' + 2}{T' - 1}. \quad (12)$$

It remains to derive relations for the corresponding expected fraction of errors, P', in terms of c and T' and in terms of N' and T'.

A message to be tested which is not a member of the set of messages, M, will be erroneously accepted as a member of M when:

(1) one of the sequence of hash addresses generated from the test message contains the same code, say C, as that generated from the test message; and

(2) that such a hash address is generated earlier in the sequence than the hash address of some empty cell.

The expected fraction of test messages, not members of M, which are erroneously accepted is then

$$P' = (\tfrac{1}{2})^{c-1}/\phi'. \quad (13)$$

Therefore

$$c = -\log_2 P' + 1 + \log_2 \frac{T' + 2}{3}, \quad (14)$$

$$N' = n \cdot \left(-\log_2 P' + 1 + \log_2 \frac{T' + 2}{3}\right) \cdot \frac{T' + 2}{T' - 1}. \quad (15)$$

Equation (15) represents the trade-offs among all three computational factors for method 1.

Analysis of Method 2

Let ϕ'' represent the expected proportion of bits in the hash area of N'' bits still set to 0 after n messages have been hash stored, where d is the number of distinct bits set to 1 for each message in the given set.

$$\phi'' = (1 - d/N'')^n. \quad (16)$$

A message not in the given set will be falsely accepted if all d bits tested are 1's. The expected fraction of test messages, not in M, which result in such errors is then

$$P'' = (1 - \phi'')^d. \quad (17)$$

Assuming $d \ll N''$, as is certainly the case, we take the log base 2 of both sides of eq. (16) and obtain approximately

$$\log_2 \phi'' = \log_e (1 - d/N'')^n \cdot \log_2 e$$
$$= -n \cdot (d/N'') \cdot \log_2 e.$$

Therefore

$$N'' = n \cdot (-\log_2 P'') \cdot \frac{\log_2 e}{\log_2 \phi'' \cdot \log_2 (1 - \phi'')}. \quad (19)$$

We now derive a relationship for the normalized time measure T''. x bits will be tested when the first $x - 1$ bits tested are 1, and the xth tested bit is 0. This occurs with probability $\phi'' \cdot (1 - \phi'')^{x-1}$. For $P'' \ll 1$ and $d \gg 1$, the approximate value of T'' (the expected value of the number of bits tested for each rejected test message) is then

$$T'' = \sum_{x=1}^{\infty} x \cdot \phi'' \cdot (1 - \phi'')^{x-1} = 1/\phi''. \quad (20)$$

Therefore

$$N'' = n \cdot (-\log_2 P'') \cdot \frac{\log_2 e}{\log_2 (1/T'') \cdot \log_2 (1 - 1/T'')}. \quad (21)$$

Equation (21) represents the trade-offs among the three computational factors for method 2.

Comparison of Methods 1 and 2

To compare the relative space/time trade-offs between methods 1 and 2, it will be useful to introduce a normalized space measure,

$$S = N/(-n \cdot \log_2 P). \quad (22)$$

S is normalized to eliminate the influence of the size of the given set of messages, n, and the allowable fraction of errors, P. Substituting the relation (22) into eqs. (15) and (21) gives

$$S' = \frac{T' + 2}{T' - 1} \cdot \left(1 + \frac{1 + \log_2 \frac{T' + 2}{3}}{-\log_2 P'}\right), \quad (23)$$

$$S'' = \frac{\log_2 e}{\log_2 (1/T'') \cdot \log_2 (1 - 1/T'')}. \quad (24)$$

We note that $S' > (T' + 2)/(T' - 1)$, and

$$\lim_{P' \to 0} S' = \frac{T' + 2}{T' - 1}. \quad (25)$$

The superiority of method 2 over method 1 can be seen directly by examining Figure 1, which is a graph of the S versus T curves for eq. (24) and the lower bound limit equation, eq. (25).[2] The curves in Figure 1 illustrate the space/time trade-offs for both methods assuming a fixed value for the expected fraction of errors, P, and a fixed number of messages, n.

[2] The reader should note that the superiority of method 2 over method 1 depends on the assumption of constant time of addressing, accessing, and testing bits in the hash area. This result may no longer hold if the effects of accessing multibit "chunks" and of calculating additional hash-coded bit addresses are taken into account.

For method 2, an increase in T'' corresponds to a decrease in ϕ'', the fraction of 0 bits in the hash field after all n messages have been hash coded. To maintain a fixed value of P, this decrease in ϕ'' corresponds to an increase in the number of bits tested per message, d, and an appropriate adjustment to the hash area size, N''. S'' is directly proportional to the hash area size N'', as shown in eq. (22). As T'' increases, and correspondingly, ϕ'' decreases, S and N'' decrease until a point of diminishing returns is reached. This point of diminishing returns is illustrated in Figure 1 where S'' is smallest for $1/T'' = 1 - 1/T''$, i.e. for $T'' = 2$.

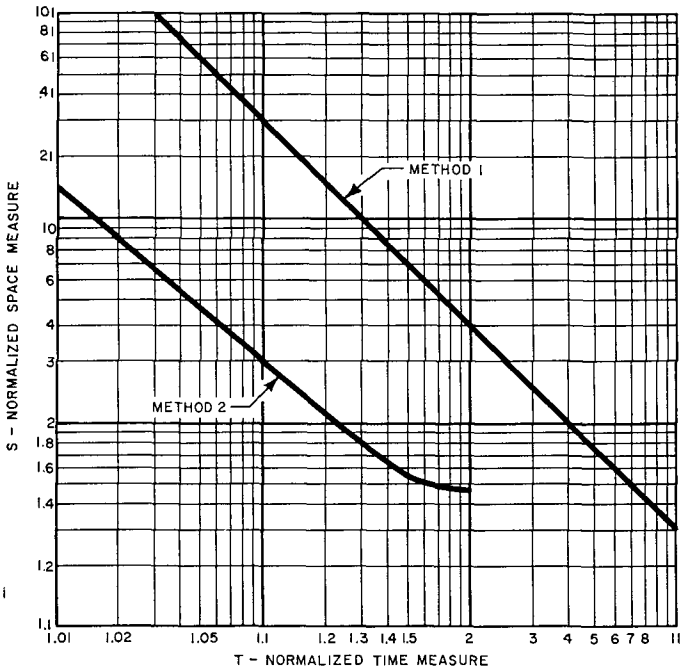

FIG. 1

Since $\phi'' = 1/T''$, this means that the smallest hash area for which method 2 can be used occurs when half the bits are 1 and half are 0. The value of S'' corresponding to $T'' = 2$ is $S'' = \log_2 e = 1.47$.

Analysis of Hyphenation Sample Application

In this section we will calculate a nominal size for a hash area for solving the sample automated hyphenation problem. Since we have already concluded that method 2 is better than method 1, we will compare the conventional method with method 2.

Let us assume that there are about 500,000 words to be hyphenated by the program and that 450,000 of these words can be hyphenated by application of a few simple rules. The other 50,000 words require reference to a dictionary. It is reasonable to estimate that at least 19 bits would, on the average, be required to represent each of these 50,000 words using a conventional hash-coding method. If we assume that a time factor of $T = 4$ is acceptable, we find from eq. (9) that the hash area would be 2,000,000 bits in size. This might very well be too large for a practical core contained hash area. By using method 2 with an allowable error frequency of, say, $P = 1/16$, and using the smallest possible hash area by having $T = 2$, we see from eq. (22) that the problem can be solved with a hash area of less than 300,000 bits, a size which would very likely be suitable for a core hash area.

TABLE I. SUMMARY OF EXPECTED PERFORMANCE OF HYPHENATION APPLICATION OF HASH CODING USING METHOD 2 FOR VARIOUS VALUES OF ALLOWABLE FRACTION OF ERRORS

P = Allowable Fraction of Errors	*N = Size of Hash Area (Bits)*	*Disk Accesses Saved*
$\frac{1}{2}$	72,800	45.0%
$\frac{1}{4}$	145,600	67.5%
$\frac{1}{8}$	218,400	78.7%
$\frac{1}{16}$	291,200	84.4%
$\frac{1}{32}$	364,000	87.2%
$\frac{1}{64}$	509,800	88.5%

With a choice for P of 1/16, an access would be required to the disk resident dictionary for approximately 50,000 + 450,000/16 $\approx$ 78,000 of the 500,000 words to be hyphenated, i.e. for approximately 16 percent of the cases. This constitutes a reduction of 84 percent in the number of disk accesses from those required in a typical conventional approach using a completely disk resident hash area and dictionary.

Table I shows how alternative choices for the value of P affect the size of the core resident hash area and the percentage of disk accesses saved as compared to the "typical conventional approach."

Acknowledgments. The author wishes to express his thanks to Mr. Oliver Selfridge for his many helpful suggestions in the writing of this paper.

RECEIVED OCTOBER, 1969; REVISED APRIL, 1970

REFERENCES

1. BATSON, A. The organization of symbol tables. *Comm. ACM 8*, 2 (Feb. 1965), 111–112.
2. MAURER, W. D. An improved hash code for scatter storage. *Comm. ACM 11*, 1 (Jan. 1968), 35–38.
3. MORRIS, R. Scatter storage techniques. *Comm. ACM 11*, 1 (Jan. 1968), 38–44.