The

TENTH ANNUAL ACM SYMPOSIUM

on

THEORY OF COMPUTING

Papers Presented at the Symposium
San Diego, California
May 1 - 3, 1978

Sponsored by the

ASSOCIATION FOR COMPUTING MACHINERY

SPECIAL INTEREST GROUP ON AUTOMATA AND COMPUTABILITY THEORY

With the Cooperation of

The IEEE Computer Society Technical Committee on
Mathematical Foundations of Computing, and

The University of California, San Diego

EXACT AND APPROXIMATE MEMBERSHIP TESTERS

Larry Carter
Robert Floyd
John Gill
George Markowsky
Mark Wegman

1. Introduction

In this paper we consider the question of how much space is needed to represent a set. Given a finite universe U and some subset V (called the vocabulary), an exact membership tester is a procedure that for each element s in U determines if s is in V. An approximate membership tester is allowed to make mistakes: we require that the membership tester correctly accepts every element of V, but we allow it to also accept a small fraction of the elements of $U - V$.

Membership testers are useful in several areas. An obvious application is spelling checking in document preparation, where many typographical errors can be recognized as not in a standard English vocabulary. Another use is in automated ciphertext-only cryptanalysis; a proposed decipherment can be discarded unless a significant fraction of the words in the decipherment are valid. In optical character recognition systems, uncertainties could be resolved in favor of words or letter sequences stored in a vocabulary of common words [RH]. Yet another application is the storing of the set of valid user IDs and passwords, or credit card numbers, or account numbers.

The first, fourth, and fifth authors are in the Automatic Programming Group at the IBM Watson Research Center, Yorktown Heights, NY 10598. The second and third authors are in the Computer Science and Electrical Engineering Departments, respectively, of Stanford University, Stanford, CA 94305. This research was supported in part by National Science Foundation Grants MCS72-03663-A04 and MCS77-07555 and by Joint Services Electronics Program Contract N00014-75-C-0601.

In this paper, lower bounds on the size of membership testers are given and algorithms are presented that nearly achieve the lower bounds. Some of these algorithms perform better than the two methods proposed by Bloom [B1] and analyzed here. If we allow an approximate membership tester to incorrectly accept a fraction 2^{-r} of the words of U, then approximately vr bits are needed for a vocabulary of v words. Although we are primarily concerned with the program's size, we are secondarily concerned with the time required to test membership. For both exact and approximate membership testers, we present theoretical procedures which nearly achieve the lower bounds and some more practical procedures which require a little extra space. For instance, one of the practical approximate membership testers can be implemented using at most $v(r+2)$ bits.

2. Lower bounds on the size of membership testers

We are motivated by applications in which V can be considered to be randomly chosen from U, with each subset of size v being equally likely. Thus, the question we ask is, "Given a set U and an integer v, what is the space required to represent any vocabulary of size v chosen from U?" If we wish to store a vocabulary that has a fair amount of structure, then there may be ways to further reduce the space requirements by taking advantage of the structure. For instance, to represent an English dictionary, we might use a Huffman (optimum variable-length) code [Hu],[Ga, pp. 52-55] to remove some of the redundancy of the English words, then partitition the set of encoded words according to lengths of the encod-

ings, and finally for each length use a representation proposed in this paper to store the sets of encodings of each length.

A standard counting argument from program size complexity [Ko],[Ch1] gives the minimal memory requirements for exact membership testers.

In this paper lg denotes logarithms base 2.

Proposition 1: In a universe of size u, at most a fraction 2^{-k} of the vocabularies of size v can be accepted by exact membership testers of size less than $\lg \binom{u}{v} - k$ bits.

Proof: There are $\binom{u}{v}$ vocabularies of v words, but only

$$2^{\lg\binom{u}{v} - k} - 1 < 2^{-k}\binom{u}{v} \qquad (1)$$

programs of size less than $\lg \binom{u}{v} - k$ bits.

Not surprisingly, as the universe size u grows, that is, as the maximum possible length of a vocabulary word grows, the memory required to represent the vocabulary increases. (The lower bound of this proposition can be increased if we assume that programs are self-delimiting [Ch2].)

We now study the storage needed to approximate vocabularies to within a specified error probability. There are two ways in which approximate membership testers can make mistakes: valid words can be unrecognized as belonging to the vocabulary (false alarms) or incorrect words can escape detection (undetected errors). It is easy to see that little memory can be saved by permitting false alarms; a membership tester with a small false alarm probability is in fact a checker for a slightly smaller vocabulary. On the other hand, if we do not insist that all incorrect words be detected, but require an undetected error probability less than 2^{-r}, then a vocabulary can be represented in an amount of storage that depends only on v and r; this storage is independent of the universe size u and for reasonable values of r is less than the program size complexity of V.

The next proposition gives a lower bound on the storage in terms of the number of vocabulary words and the undetected error probability. We assume that all incorrect words are equally likely. (We ignore for example the "local" nature of typographical errors and the consistent misspellings of certain words. These practical problems could be attacked in a spelling checking application by using a table of frequent misspellings.)

Proposition 2: In a universe of size $u >> v$, at most a fraction 2^{-k} of the vocabularies of size v can be checked with undetected error probability 2^{-r} by programs of size less than $vr - k$ bits.

Proof: Every approximate membership tester can be described by the subset W of U that it actually accepts. If the false alarm rate is 0 and the undetected error probability is 2^{-r}, then W is a superset of V that contains no more than $v+(u-v)2^{-r}$ elements. There are $\binom{u}{v}$ different vocabularies of size v, and each set W of $v+(u-v)2^{-r}$ words can correspond to approximate membership testers for no more than $\binom{v+(u-v)2^{-r}}{v}$ vocabularies. Therefore, for every v and every $r' < r$, if u is sufficiently large, then at least

$$\binom{u}{v} \Big/ \binom{v+(u-v)2^{-r}}{v} \geq 2^{vr'} \qquad (2)$$

different programs are needed to include all vocabularies of size v. The proof of the proposition is concluded by the same counting arguments as in the proof of Proposition 1: to achieve an undetected error probability of 2^{-r}, for large u, at least $vr - k$ bits are needed to represent all but 2^{-k} of the dictionaries of v words.

3. Exact membership testers

In this section, we present several methods for creating exact membership testers. We assume that, compared to the space required to store the representation of the set, the space required for the executable code of the membership tester is negligible. We also ignore any temporary work area that the procedure may require. Granting this, the first exact membership

tester achieves the lower bound of $\lceil \lg \binom{u}{v} \rceil$ bits, but it is not practical.

Exact Membership Tester 1: Assume some method of enumerating without repetition all subsets of U of size v. Represent a vocabulary V by the number of subsets that come before V in this enumeration.

The second method is more practical and requires only slightly more space.

Exact Membership Tester 2: Assume that the elements of U are the binary numbers between 0 and u-1. Let $V = \{w_1, w_2, \ldots, w_v\}$ where $w_1 < w_2 < \ldots < w_v$. Let $k = \lceil \lg \frac{u}{v} \rceil$. For each $i \leq v$, let X[i] and Y[i] be respectively the quotient and the remainder of w_i when divided by 2^k. In other words, Y[i] is the low-order k bits and X[i] is the high-order bits of w_i. If $b = \lfloor u/2^k \rfloor$ then X[i] is in {0,1,...,b-1}. We represent the set V by the array Y together with the bit string Z of v+b bits that has 1's in positions 1+X[1], 2+X[2],..., v+X[v] and 0's elsewhere.

Since the elements of X are in increasing order, the 1 we put in the (i+X[i])-th position is indeed the i-th 1 of Z. Thus the array X can be recovered from the array Z; in fact, X[i] is the number of 0's preceding the i-th 1 in Z.

Given s in U, we can determine if s belongs to V as follows:

(i) Write s as $s_1 2^k + s_2$, where $s_2 < 2^k$.

(ii) Determine j, the number of 1's before the s_1-th zero in Z, and i, the number of 1's between the s_1-th and the (s_1+1)-th zero in Z.

(iii) If k = 0, then s is not in V. Otherwise, s belongs to V iff s_2 is in {Y[j+1], Y[j+2], ..., Y[j+i]}.

Analysis: Storing V using this method requires v+b bits for Z and kv bits for Y. If we assume that u and v are powers of 2, then $k = \lg \frac{u}{v}$ and $b = \lfloor u 2^{-\lg \frac{u}{v}} \rfloor = v$. Thus this representation requires $v(\lg \frac{u}{v} + 2)$ bits. If $u >> v$ then $\lg \binom{u}{v}$, the lower bound of Proposition 1, is closely approximated by $v(\lg \frac{u}{v} + \lg e)$, and $\lg e \approx 1.44$.

Step (ii) requires examining a large portion of the long bit string Z. If we apply our membership tester to a sorted list of test words, then this may not matter. Otherwise, we can speed up the process by precomputing and storing the number j of step (ii) for selected values of s_1.

Exact Membership Tester 3: Let t be an integer with $1 \leq t \leq v$. The smaller t is, the faster the membership tester will operate and the more space will be required. The representation of V consists of the array Y and the bit string Z of Membership Tester 2, along with the array W where W[i] is the number of 1's before the (i×t)-th zero of Z.

Step (ii) can now be replaced by the following faster sequence of steps:

(iia) Divide s_1 by t to obtain q and r such that $s_1 = qt+r$ and $r < t$. We know that in the initial W[q] + qt bits of Z, there are exactly qt zeroes and W[q] 1's.

(iib) In the substring of Z starting at the (W[q]+qt+1)-th bit, count how many 1's there are before the r-th zero. Add this number to W[q] to obtain j. As before, let k be the number of 1's between the r-th and the (r+1)-th zero.

Analysis: Assuming that u and v are powers of 2, Exact Membership Tester 3 requires

$$v(2 + \lg\tfrac{u}{v}) + \tfrac{v}{t} \lg 2v = v(2 + \lg\tfrac{u}{v} + \tfrac{1+\lg v}{t}) \quad (3)$$

bits of storage. The time required to determine if s is a member of V is, for the "average" s, a constant plus the time required to count the number of 1's in a string of t bits. The choice of the parameter t should be motivated by the particular application, but it may help to notice that by halving t, the number of extra bits is doubled and the speed is roughly cut in half.

We conclude this section with two observations.

First, another operation that one might wish to perform on a set is to output the i-th largest member. The representation of Exact Membership Tester 3 can be used for that operation, but it turns out to be a little faster if the number of 0's before the $(i \times t)$-th 1 is stored instead of (or in addition to) the number of 1's before the $(i \times t)$-th 0.

Second, we note that there is an even more compact (and more complicated) practical exact membership tester. Briefly, some additional space can be saved by choosing the parameter k of Exact Membership Tester 2 to be smaller than $\lg \frac{u}{v}$, and using a Huffman code to reduce the length of the bit string Z which now contains more 0's than 1's. For instance, if we choose k to be $\lg \frac{u}{v} - 1$ and encode groups of 3 bits, then we reduce the $v(\lg \frac{u}{v} + 2)$ bits required by Exact Membership Tester 2 to $v(\lg \frac{u}{v} + 1.81)$ bits.

4. Approximate membership testers

In this section we present several methods for creating approximate membership testers for a vocabulary V. Each tester has an associated preprocessor that digests V and produces a compressed representation of a superset W of V. The input to the tester is the compressed vocabulary and a possible word s; the membership tester estimates whether s is in V by actually computing if s is in W. Under the assumption that all errors are equally likely, the undetected error probability is $(w-v)/(u-v)$, where w is the size of W.

In each procedure we seek to achieve an undetected error probability of 2^{-r}. Thus the design parameters are v, the number of vocabulary words and r, the reliability exponent. The universe size u does not appear explicitly in the description of these procedures, but does affect the choice of the necessary hash functions.

The first two approximate membership testers are essentially the same as hashing schemes proposed in [B1]. We have analyzed these methods in order to allow them to be compared against the third tester. In each case, we have adjusted the parameters of these methods to achieve an undetected error probability of 2^{-r}.

Preprocessor 1: Let $h_1, h_2, \ldots, h_r$ be r "independent" hash functions such that $h_j : U \rightarrow \{1, 2, \ldots, (\lg e)vr\}$. Prepare a hash table of $(\lg e)vr$ bits as follows. Initialize the table to zeroes. Enter each vocabulary word v_i into the table by setting the bits in the r locations $\{h_j(v_i): j=1,2,\ldots,r\}$. (Note that certain bits in the table may be set on account of more than one vocabulary word.)

Approximate Membership Tester 1: Given a test word s, test the bits in the dictionary hash table at locations $h_1(s), h_2(s), \ldots, h_r(s)$. An error is detected iff any one of these bits is not set.

Analysis: At most vr bits of the hash table are set by the preprocessor, but in fact because of duplications the expected number of bits set can be calculated to be $\frac{1}{2}(\lg e)vr$. (This holds under the generous assumption that the r hash functions operating on the v vocabulary words produce independent addresses.) Thus, on the average, only half of the bits in the table are set, and so the undetected error probability, that is, the probability that for a randomly chosen word s all bits in locations $h_1(s), h_2(s), \ldots, h_r(s)$ are set, is at most about 2^{-r}.

This tester requires r probes into the table to verify correct words, but only an average of about 2 probes to detect incorrect words. This method can be modified, at only a slight increase in the undetected error probability, so that for each word s, the hash values $\{h_j(s)\}$ are in the same region in memory, thereby allowing for efficient implementation of this method in paged memories.

The basic idea of the second approximate membership tester is to store a hash function of each vocabulary word in a hash table. If the length of the stored hash value and the size of the table are carefully chosen, then a reasonably efficient and compact representation of the vocabu-

lary is obtained. To simplify the description, we suppose that r is a power of 2.

Preprocessor 2: Prepare a hash table of av entries, where

$$a = 1 + \frac{1}{r + \lg r} \qquad (4)$$

and each table entry is $r + \lg r$ bits in size. The total storage of the table is $v(r + \lg r + 1)$ bits. Let h be an $(r + \lg r)$-bit hash function that excludes the value 0, and let $h_1, h_2, h_3, \ldots$ be an "independent" sequence of hash functions mapping U into $\{1, 2, \ldots, av\}$. The hash table is initially empty, that is, all entries are zero. Enter the value $h(v_i)$ into the hash table for each vocabulary word v_i; a free location in the table is found by checking locations $h_1(v_i), h_2(v_i), h_3(v_i), \ldots$ until an index j is found for which the entry in location $h_j(v_i)$ is 0.

Approximate Membership Tester 2: Given a test word s, compute $h(s)$ and compare with the entries in locations $h_1(s), h_2(s), h_3(s), \ldots$ until j is found such that the entry in location $h_j(s)$ equals 0 or $h(s)$. In the former case an error has been detected, while in the latter case s is assumed to be a valid word. (If desired, j can be constrained to be no more than the largest number of hashes needed to enter any word in the vocabulary preprocessing; with high probability $j \leq r \ln r$.)

Analysis: We argue informally; the inequalities can be justified with a little calculation. The average number of probes into the table for a random word s before finding a vacant location is $r + \lg r + 1$. Therefore the undetected error probability is no more than about

$$\frac{r + \lg r + 1}{2^{r+\lg r} - 1} \approx \frac{r + \lg r + 1}{r2^r} \approx 2^{-r}. \qquad (5)$$

On the average, about r probes are required to detect errors, but only $O(\ln r)$ probes are required for correct words. (This holds under the pessimistic assumption that all correct words are equally likely; hashing the most frequently occurring vocabulary words first when preparing the compressed dictionary will result in a smaller number of probes for correct words.)

The second tester uses less storage for large r than the previous method, but for interesting values of r, say 10, the two methods are comparable. One application of this second method might be in compiling programs with very long identifiers. By storing the first few symbols of the identifier together with a hash function of the entire identifier, the compiler can distinguish with high probability long identifiers that agree in most positions.

The third tester is similar to the second in that the vocabulary is represented by a set of values of the words produced by a hash function h. However, instead of storing approximately r-bit values in a hash table, we enter $r + \lg v$ bits into a linear array; the tester compares a test word's hash value against every value in this table.

The intuition behind this method is that we hash our vocabulary into $v2^r$ bins. Then the probability that an invalid word will collide with a vocabulary word is no more than 2^{-r}. To simplify the description below, we assume that the vocabulary size v is a power of 2.

Preprocessor 3: Let H be a universal$_2$ class of hash functions from U to $N = \{0, 1, \ldots, v2^r - 1\}$. Choose h at random from H. Let $V = \{w_1, w_2, \ldots, w_v\}$ and let M be the set $\{h(w_i): i = 1, 2, \ldots v\}$. Create an exact membership tester for M using the second or third method of the previous section. Since $|M| \leq v$ (we assume for simplicity that v is a power of 2), this requires at most

$$|M| \left\lceil \lg \frac{|N|}{|M|} + 2 \right\rceil \leq v(r + 2) \qquad (6)$$

bits when using Exact Membership Tester 2. Faster execution can be achieved at the expense of using more space by Exact Membership Tester 3.

Approximate Membership Tester 3: Given a test word s, accept s iff $h(s)$ belongs to M.

Analysis: If the hash function h

produces values distributed uniformly over N, then the fraction of words in U that hash to values in $\{h(w_i): i=1,\ldots,v\}$ is at most $|V|/|N|$. Since h was chosen randomly from a universal$_2$ class of hash functions [CW], we can invoke Proposition 2 of [CW] to conclude that if s is not in V then the probability that $h(s)$ belongs to M is at most

$$\frac{|V|}{|N|} = \frac{v}{v2^r} = 2^{-r}. \qquad (7)$$

The average performance (in the sense of [Gi] or [Ra]) of this tester is independent of the particular vocabulary V. For all vocabularies of a given size, Approximate Membership Tester 3 requires the same amount of space. We can almost claim that the error probability and the time to check an element are also vocabulary-independent. This claim rests on the fact that the estimates of speed and accuracy apply to any one vocabulary if we average over the choice of hash functions.

A significant advantage of the third tester is that it does not assume the existence of an independent sequence of hash functions, but requires only a function that hashes U nearly uniformly onto N. The third tester also compares quite favorably to the first two in execution time. To make the comparison, we assume that the time required to count the number of 1's in a byte is roughly equal to the time required to hash a byte. Then the parameter t of Exact Membership Tester 3 should be chosen so that Approximate Membership Tester 3 takes about the same amount of time as the membership tester being compared against. It turns out that for $r \geq 2$, Approximate Membership Tester 3 requires less space than Approximate Membership Tester 1. Neither the second nor the third membership tester is clearly superior for $r \leq 3$, but for all $r > 3$, the third tester uses less space than the second.

Finally, we present a space-optimal (but absurdly impractical) approximate membership tester.

Preprocessor 4: For $m = 1,2,3,\ldots$, generate in some fixed order all collections $C = \{W_1, W_2, \ldots, W_m\}$ where each W_i is a subset of U of size $\lfloor u2^{-r} \rfloor$. For each C, the preprocessor tests if every v-element subset of U is contained in some W_i. This phase of the preprocessor is finished when the first satisfactory C is found. Now the encoding used for the vocabulary V is simply the smallest i such that W_i contains V.

Approximate Membership Tester 4: The membership tester duplicates all the work involved in constructing the first satisfactory C. Then it checks if the test word s belongs to W_i.

5. Some related combinatorial results

Erdős and Spencer [ES] define $M(u,k,v)$ to be the smallest number of k-element subsets of a universe U of u elements such that every v-element subset of U is contained in one of the k-element subsets. The problem we have been considering is for each v-element subset of U, to find an approximate membership tester that accepts some superset W of V, where $|W| \approx u2^{-r}$. Thus the number of bits needed to specify an approximate membership tester for any v-element vocabulary bounds above $\lg M(u,u2^{-r},v)$. By using Approximate Membership Tester 3 with Exact Membership Tester 1, we derive $\lg M(u,u2^{-r},v) \leq \lg \binom{v2^r}{v}$. In other words,

$$\lg M(u,k,v) \leq \lg \binom{uv/k}{v} \approx v \lg \frac{eu}{k} \qquad (8)$$

where e is the base of the natural logarithm.

On the other hand, the upper bound that Erdős and Spencer derive (using a probabilistic and nonconstructive argument) is

$$M(u,k,v) \leq (1 + \ln \binom{k}{v}) \binom{u}{v} / \binom{k}{v}. \qquad (9)$$

From this it follows that

$$\lg M(u,k,v) \leq v \lg \frac{u}{k} + \lg \ln \binom{k}{v}. \qquad (10)$$

Which of these two upper bounds on $\lg M(u,k,v)$ is better depends on the particular values of u, v, and r, but in either case we learn something.

When $1.44v < \lg \ln \binom{k}{v}$, then we

have an improved upper bound on $\lg M(u,k,v)$, namely $v \lg \frac{eu}{k}$. This result is not of tremendous combinatorial interest, since k must be on the order of 2^{2^v} for the new bound to be smaller than that given by Erdős and Spencer, but it does give a better asymptotic result. Specifically, if we hold v constant and let u and k go to infinity in a fixed ratio, then $v \lg \frac{eu}{k}$ remains constant while the Erdős-Spencer bound slowly grows to infinity.

Otherwise, when $1.44v > \lg \ln \binom{k}{v}$, we know that Approximate Membership Tester 3 is not optimal. For example, suppose that U is the set of strings over the Roman alphabet of length 100 (so that $u = 26^{100}$), and that $v = 100{,}000$ and $r = 10$. Then it turns out that Approximate Membership Tester 3 requires 1,200,000 bits, but Approximate Membership Tester 4 needs somewhere between 1,000,000 and 1,000,025 bits.

Acknowledgments

This research was supported in part by National Science Foundation Grants MCS72-03663-A04 and MCS77-07555 and by Joint Services Electronics Program Contract N00014-75-C-0601.

References

[Bl] B.H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," Comm. ACM 13 (1970), pp. 422-426.

[CW] J.L. Carter & M.N. Wegman, "Universal Classes of Hash Functions," Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, May, 1977, pp. 106-112.

[Ch1] G.J. Chaitin, "On the Length of Programs for Computing Finite Binary Sequences," Journal Assoc. Comput. Mach. 13 (1966), pp. 547-569.

[Ch2] G.J. Chaitin, "A Theory of Program Size Formally Identical to Information Theory," Journal Assoc. Comput. Mach. 22 (1975), pp. 329-340.

[ES] P. Erdős and J. Spencer, Probabilistic Methods in Combinatorics, Chapter 13, Academic Press, New York, 1974.

[Ga] R.G. Gallager, Information Theory and Reliable Communication, Wiley, New York, 1968.

[Gi] J. Gill, "Computational Complexity of Probabilistic Turing Machines," SIAM Journal on Computing 6 (1977), pp. 675-695.

[Hu] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," Proc. IRE 40 (1952), pp. 1098-1101.

[Ko] A.N. Kolmogorov, "Three Approaches to the Definition of the Concept 'Quantity of Information'," Probl. Pederachi Inform. 1 (1965), pp. 3-11.

[Ra] M.O. Rabin, "Probabilistic Algorithms," Algorithms and Complexity: New Directions and Recent Results, J.F. Traub, ed., Academic Press, New York, 1976, pp. 21-40.

[RH] W.S. Rosenbaum and J.J. Hilliard, "Multifont OCR Postprocessing System," IBM Journal of Research and Development 19 (1975), pp. 398-421.