Efficient Storage and Retrieval by Content and Address of Static Files

PETER ELIAS

Massachusetts Institute of Technology, Cambridge, Massachusetts

ABSTRACT. We consider a set of static files or inventories, each consisting of the same number of entries, each entry a binary word of the same fixed length selected (with replacement) from the set of all binary sequences of that length, and the entries in each file sorted into lexical order. We also consider several retrieval questions of interest for each such file. One is to find the value of the jth entry, another to find the number of entries of value less than k.

When a binary representation of such a file is stored in computer memory and an algorithm or machine which knows only the file parameters (i.e. number of entries, number of possible values per entry) accesses some of the stored bits to answer a retrieval question, the *number of bits stored* and the *number of bits accessed per retrieval question* are two cost measures for the storage and retrieval task which have been used by Minsky and Papert. Bits stored depends on the representation chosen: bits accessed also depends on the retrieval question asked and on the algorithm used.

We give firm lower bounds to minimax measures of bits stored and bits accessed for each of four retrieval questions, and construct representations and algorithms for a bit-addressable machine which come within factors of two or three of attaining all four bounds at once for files of any size. All four factors approach one for large enough files.

KEY WORDS AND PHRASES: file, storage, retrieval, access, inverted file, efficiency, computational complexity

CR CATEGORIES: 3.70, 3.72, 3.74, 5.25, 5.6

1. *Tables, Files, and Inventories*

Consider a collection of N words, each of length w binary digits. If the jth word is interpreted as the w-bit binary representation of an integer x_j and the words are given in a specified order the collection may be taken to represent a *table*, as in the table of values of an arbitrary function from the integers $[1, N]$ to the integers $[0, W]$, where $W = 2^w - 1$. Denote by $T(N, W)$ the set $[0, W]^N$ of all possible N-entry, w-bit tables.

$$\mathbf{x} = (x_1, x_2, \cdots, x_N) \in T(N, W) \quad \text{iff} \quad x_j \in [0, W], \quad j \in [1, N]. \tag{1}$$

If the original collection of N words is not ordered it is common practice to sort the words into lexical order before storing them in memory so as to make access easier. The result may be taken to represent the sequence of values of a (perhaps weakly) monotone increasing function from $[1, N]$ to $[0, W]$. An example is the ordered set of w-bit stock-numbers on N items in an inventory. Let $I(N, W)$ denote the set of such *inventories*:

$$\mathbf{x} \in I(N, W) \quad \text{iff} \quad \mathbf{x} \in T(N, W) \quad \text{and} \quad 1 \leq i < j \leq N \Rightarrow x_i \leq x_j. \tag{2}$$

If the collection has no duplicates it may be taken to represent a *file*. The class $F(N, W)$

This research was partially supported by the Joint Services Electronics Program (Contract DA28-043-AMC-02536(E)).

Author's address: Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139.

of files is the subset of $I(N, W)$ which is strictly increasing: $\mathbf{x} \in F(N, W)$ iff it satisfies (2) with $x_i \leq x_j$ replaced by $x_i < x_j$. We do not deal with files per se here, but with the more general class of inventories (see, however, [2]).

Given a representation of $\mathbf{x} \in I(N, W)$ as a string of bits stored in a binary bit-addressable memory (for example the canonical representation of $\mathbf{x}$ as the Nw-bit concatenation of the N original entries of w bits each, concatenated in order of increasing value), it requires a certain amount of effort for an algorithm which only knows the parameters N and w to answer a question about $\mathbf{x}$ by accessing some of the bits of the stored representation. The effort required depends both on what representation is chosen and on what retrieval question is asked.

Given an inventory (or file) $\mathbf{x} \in I(N, W)$, Minsky and Pappert [7] use two measures of performance for a binary representation of $\mathbf{x}$ and an associated retrieval algorithm. One is the *total number of bits of memory needed* to store the representation. The other is the *average number of those bits* which must be accessed by the algorithm in order to answer a retrieval question. They consider two kinds of retrieval questions, the exact match question "Is there a $j \in [1, N]$ with $x_j = k$?" and the approximate match question "For which j does the w-bit binary representation of x_j differ from the w-bit binary representation of k in fewest bit positions?", both $k \in [0, W]$. They explore the trading relations between their two cost measures for these two questions using a variety of representations. They find a representation and an algorithm which have both storage and access costs of only a few times the minimum possible values for the exact match question. Approximate match seems to take either very large storage or a great deal of access.

Fano [3] gives a representation and an algorithm for an exact-match-and-address question "Is there a $j \in [1, N]$ such that $x_j = k$ and if so, what is it?" His representation is essentially equivalent to the one derived independently by the author, presented in [2] and in Section 3. He does not consider the number of bits accessed per question.

We use the same kinds of costs as Minsky and Papert and get results from four elementary retrieval questions, one of which is closely related to Fano's.

The first question "What is $\mathbf{x}$?" is the *identity* or *archival* question. Its answer is a printout of the inventory $\mathbf{x}$ in canonical form, i.e. the Nw-bit sequence of the w-bit representations of the N entries of $\mathbf{x}$, printed in increasing order.

The second question "What is x_j?" for $j \in [1, N]$ is the *direct* or *table-lookup* question. Its answer is the integer x_j, or its w-bit binary representation. It is the natural question to ask of a table. For an inventory, the direct question asks for the stocknumber on the item in the jth of N bins, or the file of a customer whose file number is j, and is less natural.

The questions of greatest interest for inventories and files are of an *inverse* character. "Do we have any items in stock with stocknumber k? If so, how many? And in which bins?" for $j \in [0, W]$. Our third question is "For how many $j \in [1, N]$ is $x_i < k$?", $k \in [1, W]$, the *inverse* question. We denote its integer answer by $\bar{x}_k \in [0, N]$.

A nondecreasing function from $[1, N]$ to $[0, W]$ does not usually have an inverse which is a function. The inverse image of a value $k \in [0, W]$ is an interval (often empty when $W \gg N$) of adjacent integers in $[1, N]$, where the entries of value k are located. In terms of answers to the inverse question, $\bar{x}_{k+1}$ is the largest integer in that interval and $\bar{x}_k + 1$ is the smallest. If $\bar{x}_k = \bar{x}_{k+1}$ the interval is empty, and the number to the exact match question of Minsky and Papert is "no."

Given any set S, let the magnitude $|S|$ denote the number of its members. Then formally

$$\bar{x}_k = |\{j \in [1, N] \mid x_j < k\}|, \qquad \bar{x}_{k-1} - \bar{x}_k = |\{j \in [1, N] \mid k \leq x_j < k + 1\}| = |\{j \in [1, N] \mid x_j = k\}|. \tag{3}$$

Since $\bar{x}_k$ is the magnitude of a set which grows with k, the N-tuple $\bar{x} = (\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_W) \in [0, N]^W$ is monotone: $1 \leq i \leq j < W \Rightarrow 0 \leq \bar{x}_i \leq \bar{x}_j \leq N$, so (2) is satisfied and $\bar{\mathbf{x}} \in I(W, N)$ is an inventory called the *complement* of $x \in I(N, W)$.

Complementation is self-inverse. Let $\mathbf{y} = \bar{\mathbf{x}} \in I(W, N)$. Then $\bar{y}_j$ is the size of the set S_j, where

$$S_j = \{k \in [1, W] \mid y_k < j\} = \{k \in [1, W] \mid \bar{x}_k < j\} = \{k \in [1, W] \mid x_j \geq k\},$$
$$\bar{y}_j = |S_j| = x_j, \quad (4)$$

by the definition (3) of complementation. Thus complementation provides a 1-1 map from $I(N, W)$ onto $I(W, N)$.

Direct and inverse questions for $\mathbf{x} \in I(N, W)$ are respectively inverse and direct questions for $\bar{\mathbf{x}} \in I(W, N)$. A binary representation of $\mathbf{x}$ and its associated algorithms for direct and inverse questions can therefore be used as a representation of $\bar{\mathbf{x}}$ by interchanging the names of the algorithms. Because of this duality it is necessary to construct representations and algorithms only for $W \geq N$.

To permit symmetric treatment of $\mathbf{x}$ and $\bar{\mathbf{x}}$ requires a canonical representation of $\bar{\mathbf{x}}$ as an ordered set of W strings, each containing the same number n of bits. This in turn requires that there be an integer n with

$$N = 2^n - 1, \qquad W = 2^w - 1, \qquad w \geq n \geq 1, \quad (5)$$

which we assume henceforth (except for the proof of Theorem 1). Our fourth retrieval question asks "What is $\bar{\mathbf{x}}$?" and its answer is the nW-bit canonical representation of $\bar{\mathbf{x}}$: it is introduced by the symmetry of the analysis.

2. *Measures of Performance and Results*

Let each inventory $\mathbf{x} \in I(N, W)$ be represented by a distinct finite binary string whose length $L(\mathbf{x})$ may vary with $\mathbf{x}$. The integer $L(\mathbf{x})$ is called the storage cost of the representation of $\mathbf{x}$, and the resulting map from $I(N, W)$ into the finite binary strings $\{0, 1\}^*$ is called a *representation* of the set $I(N, W)$.

Let a binary bit-addressable memory $\mathbf{m}$ store the representation of $\mathbf{x}$ in its first $L(\mathbf{x})$ bits, leaving its later bits free to take values set by other users or programs. Associated with the representation of $I(N, W)$ is a set of four retrieval algorithms or machines which have available the parameters N and W and can access the bits of $\mathbf{m}$ to answer the direct and archival questions about both $\mathbf{x}$ and $\bar{\mathbf{x}}$. Each algorithm has an access cost per use given by the number of bits it must access in $\mathbf{m}$ to find the answer to its questions. $A_d(\mathbf{x}, j)$ and $A_i(\mathbf{x}, k)$ are the access costs of the direct and inverse algorithms when finding x_j and $\bar{x}_k$, respectively, for $\mathbf{x} \in I(N, W)$, $j \in [1, N]$, $k \in [1, W]$: $A_{da}(\mathbf{x})$ and and $A_{ia}(\mathbf{x})$ are the access costs of the archival algorithms finding $\mathbf{x}$ and $\bar{\mathbf{x}}$.

The storage cost and the four access costs may each differ for different $\mathbf{x} \in I(N, W)$, and $A_d(\mathbf{x}, j)$ and $A_i(\mathbf{x}, k)$ may also vary with j and k. Since the analysis deals with static files, it applies to a user who keeps the same $\mathbf{m}$ (representing the same $\mathbf{x} \in I(N, W)$) a long time and uses it to answer many questions. Such a user can average over questions but *not* over inventories. Thus let

$$A_d(\mathbf{x}) = (1/N) \sum_{j=1}^{N} A_d(\mathbf{x}, j), \qquad A_i(\mathbf{x}) = (1/N) \sum_{k=1}^{W} A_i(\mathbf{x}, k). \quad (6)$$

We measure the performance of a representation and its associated algorithms by the five numbers L, A_d, A_i, A_{da}, A_{ia}, each of which is the *maximum* over $\mathbf{x} \in I(N, W)$ of the corresponding function:

$$L = \max_{\mathbf{x} \in I} L(\mathbf{x}), \quad A_d = \max_{\mathbf{x} \in I} A_d(\mathbf{x}), \quad A_i = \max_{\mathbf{x} \in I} A_i(\mathbf{x}),$$
$$A_{da} = \max_{\mathbf{x} \in I} A_d(\mathbf{x}), \quad A_{ia} = \max_{\mathbf{x} \in I} A_i(\mathbf{x}). \quad (7)$$

These maximum measures can be guaranteed to any customer given his values of N and W but not the contents of his file.

We find lower bounds to each of these five cost measures, which hold for all representations and algorithms, and summarize the results in Theorem 1.

We first bound below the storage L, which is the length of the longest string in the representation of $I(N, W)$. For any $\mathbf{x}$ whose representation takes $L(\mathbf{x}) < L$ bits, the algorithms must all work when the $L - L(\mathbf{x})$ bits in $\mathbf{m}$ following the first $L(\mathbf{x})$ bits are padded out with 0's (which might belong to some other data base). The padded-out strings of uniform length L must include at least one distinct string for each $\mathbf{x} \in I(N, W)$ since no two inventories give the same answer to all questions. Thus 2^L must be at least as great as the number $|I(N, W)|$ of possible inventories.

Next we use an information-theoretic argument to lower bound the number of accesses required to answer a retrieval question. Suppose (for the sake of this lower bounding argument only) that a probability $P(\mathbf{x})$ has been assigned to each $\mathbf{x} \in I(N, W)$. A retrieval algorithm accesses $\mathbf{m}$ and for each $\mathbf{x} \in I(N, W)$ receives some sequence of values of the accessed bits. If it prints the correct answer to the retrieval question and halts no matter which $\mathbf{x}$ has its representation stored in $\mathbf{m}$, then by the converse to the coding theorem of information theory the average number of bits it receives cannot be less than the entropy (in bits) of the set of answers to the question. If for a given retrieval question it is possible to choose $P(\mathbf{x})$ so as to make all of the possible answers to that question equiprobable, then the entropy of the set of answers for that $P(\mathbf{x})$ is just the logarithm (base 2) of the number of different possible answers. (See, e.g., Fano [4, p. 63] and Gallager [5, p. 51].)

For each of the four retrieval questions it is possible to find a probability assignment which makes all answers equiprobable. The assignment P_a will do for the two archival questions, where $P_a(\mathbf{x}) = 1/|I(N, W)|$ for all $\mathbf{x} \in I(N, W)$. The assignment P_d gives equal probability to the $W + 1$ possible answers $0, 1, \cdots, W$ to the direct question, and P_i gives equal probability to the $N + 1$ possible answers $0, 1, \cdots, N$ to the inverse question, where

$$P_d(\mathbf{x}) = \begin{cases} 1/(W+1) \text{ if } \mathbf{x} \text{ is constant, } x_j = c,\ j \in [1, N], \\ 0 \text{ otherwise;} \end{cases}$$

$$P_i(\mathbf{x}) = \begin{cases} 1/(N+1) \text{ if } \bar{\mathbf{x}} \text{ is constant, } x_k = c,\ k \in [1, W], \\ 0 \text{ otherwise.} \end{cases}$$

Thus for the archival questions

$$\sum_{x \in I} P_a(\mathbf{x}) A_{da}(\mathbf{x}) \geq \log |I(N, W)|, \qquad \sum_{x \in I} P_a(\mathbf{x}) A_{ia}(\mathbf{x}) \geq \log |I(N, W)|. \tag{8}$$

And for the direct and inverse questions

$$\begin{aligned} &\sum_{x \in I} P_a(\mathbf{x}) A_d(\mathbf{x}, j) \geq \log (W+1) \text{ for all } j \in [1, N], \\ &\sum_{x \in I} P_i(\mathbf{x}) A_i(\mathbf{x}, k) \geq \log (N+1) \text{ for all } k \in (1, W). \end{aligned} \tag{9}$$

Averaging (9) further over $j \in [1, N]$ and $k \in [1, W]$ with equal weights gives

$$\sum_{x \in I} P_d(\mathbf{x}) A_d(\mathbf{x}) \geq \log (W+1), \qquad \sum_{x \in I} P_i(\mathbf{x}) A_i(\mathbf{x}) \geq \log (N+1). \tag{10}$$

Using for (8) and (10) the fact that the largest number in a set is no smaller than the average, and for (8) the fact that the largest number in a set of integers is an integer, completes the proof of

THEOREM 1. *Let $I(N, W)$ be the weakly increasing subset of N-tuples in $[0, W]^N$. Let ρ be a binary relation from $I(N, W)$ onto $\{0, 1\}^L$ whose converse ρ^{-1} is a function from $\{0, 1\}^L$ onto I. Let the measures of performance of a set of four algorithms which find the right answers to the direct, inverse, direct archival, and inverse archival questions about x by accessing bits of*

TABLE I

Type of Representation	Total Bits Stored, L	Average Bits Accessed Per Question			
		A_d	A_i	A_{da}	A_{ia}
Canonical for $\mathbf{x}$	Nw	$\mathbf{w}$	nw	Nw	Nw
Canonical for $\bar{\mathbf{x}}$	nW	nw	$\mathbf{n}$	nW	nW
Enumeration of I	$\mathbf{L_0}$	L_0	L_0	$\mathbf{L_0}$	$\mathbf{L_0}$

the image of x under ρ be defined by (7). *Then*

$$L \geq L_0 = \lceil \log | I(N, W) | \rceil, \qquad A_d \geq \log(W+1) = w, \qquad A_i \geq \log(N+1) = n,$$
$$A_{da} \geq L_0, \qquad A_{ia} \geq L_0.$$

Each of the five bounds in Theorem 1 is attained for some algorithm and some representation, as shown in boldface in Table I. The canonical representation of $\mathbf{x}$ as the concatenation of w-bit representations of its N entries in order of increasing value permits finding x_j by accessing just the minimal w bits at addresses $(j-1)w + t$, $t \in [1, w]$. A similar table lookup gives $\bar{x}_k$ in the minimal n accesses to memory if the nW-bit canonical representation of $\bar{\mathbf{x}}$ is stored. Any enumeration of I by a 1-1 map from $I(N, W)$ onto the (L_0-bit binary representations of the) integers in $[0, | I(N, W) | -1]$ takes minimum storage. An algorithm which reads all L_0 bits into working memory and computes or has stored the inverse of the map can answer all four kinds of questions with L_0 accesses to memory, which is minimal for the archival questions.

None of the representations in Table I can attain or approximate all five minima at once, using any set of retrieval algorithms. One might in fact anticipate that a representation which uses near to minimal storage will necessarily have an algorithm which must make much more than the minimal number of accesses to answer a direct or inverse question. The bulk of this paper is devoted to the construction of representations and algorithms which show that such an anticipation is not correct, by approximating all five minima using a single representation and set of retrieval algorithms. We summarize the results below in Theorems 2 and 3. Theorem 3 gives the general results for arbitrary n and w with $N = 2^n - 1$, $W = 2^w - 1$. Theorem 2 deals with the special case $n = w$, which is proved first and then used in the proof of Theorem 3.

THEOREM 2. *Let $w = n \geq 1$. Then*

(*i*) $L_0 = \lceil 2N - (n+1)/2 \rceil$,

and for each n there is a binary representation of $I(N, N)$, four associated algorithms, and an ϵ_n, $2^{\frac{1}{2}} > \epsilon_n \geq 0$, with

(*ii*) $L \leq L_0(1+\epsilon_n)$, $A_d = A_i \leq n(1+\epsilon_n)$, $A_{da} = A_{ia} = 2N \leq L_0(1+\epsilon_n)$.

For integer $k \geq 0$ and $n = 2^k$, $\epsilon_n \leq 1$. And

(*iii*) $\lim_{n\to\infty} \epsilon_n = 0$.

There is also a representation and two algorithms, for answering the two archival questions only, with

(*iv*) $L = A_{da} = A_{ia} = 2N \leq L_0(1 + n/2N)$.

THEOREM 3. *Let $n \geq 1$, $w - n = s \geq 0$. Then*

(*i*) $L_0 > N(s + (1+2^s)\log(1+2^{-s})) - 1 - n/2 > N(s + \log e) - 1 - n/2$,

and there is a binary representation of $I(N, W)$, four associated algorithms and an ϵ_n, $2^{\frac{1}{2}} > \epsilon_n > 0$, with

(*ii*) $L \leq N(s + 2(1+\epsilon_n)) - 2(n - \epsilon_n)$, $A_{da} = A_{ia} \leq N(s+2)$, $A_d \leq w(1+\epsilon_n)$,
$A_i \leq n(1+\epsilon_n) + 2(1-2^{-s}) \leq n(1+\epsilon_n) + 2$.

For integer $k \geq 0$ *and* $n = 2^k$, $\epsilon_n \leq 1$. *And*

(*iii*) $\lim_{n \to \infty} \epsilon_n = 0$.

There is also a representation and two algorithms, for answering the two archival questions only, with $L = A_{da} = A_{ia} = N(s + 2)$.

Theorems 1 and 3 together provide very tight bounds on five measures of the computational complexity of a specific storage and retrieval task, and show that there are systems whose performance has nearly minimal cost by all five measures. However these five measures are not a complete set. In addition to the bits stored and the bits accessed per retrieval question, the *bits of state information* required by the algorithms or machines which access the representation and answer the question are also a cost. So is the *amount of combinational logic or read-only memory* which the algorithms or machines require.

We do not compute such additional measures here, but note that the algorithms and representations constructed below are nearly minimal in such respects also. A number of state bits sufficient to store an argument or two (n bits each), a value or two (w bits each) and an address or two ($\log L$ bits each) will do, together with a few n-bit or w-bit parameters stored in read-only memory. And the number of operations required (additions and subtractions of 1, shifts of 1 bit-position, bit-compares, etc.) is only a few per accessed bit.

For the canonical representations of $\mathbf{x}$ and $\tilde{\mathbf{x}}$, the associated table lookup and logarithmic search algorithms which give the first two lines of Table I also have such minimal complexity. (So does the improved bit-by-bit logarithmic search suggested by Minsky and Papert [7].) However, the enumeration representation seems to require more complex algorithms, which perform arithmetic with L_0-bit numbers (see Lehmer [6], Schalkwijk [8], Cover [1].)

The algorithms and representations developed below to prove Theorems 2 and 3 require a bit-addressable memory. A machine which is constrained to access a word of many bits can do no better, as measured in bit-accesses to memory, and may do worse, so Theorem 1 still applies but not the guarantees in Theorems 2 and 3.

3. *Quotients, Remainders, and Unary Representations*

To develop the representations needed to prove Theorems 2 and 3, it is convenient to shift at this point from the N-tuples $\mathbf{x} \in I(N, W)$ to the bit strings of length Nw which are their canonical representations. This section introduces notation for such strings, gives a 1-1 map from $I(N, W)$ into $I(N, N) \times T(N, (W + 1)/(N + 1) - 1)$ and an economical unary representation of $I(N, N)$ equivalent to Fano's [3], and finds tight bounds on L_0. Section 4 gives retrieval algorithms for the representation of $I(N, N)$, Section 5 constructs directories for use by the algorithms and proves Theorem 2, and Section 6 returns to the general case and proves Theorem 3.

Let f be a finite binary string in $\{0, 1\}^*$. Let $|f|$ denote the length of f, $f(i)$ the value of its ith bit, and $\|f\|$ the integer of which f is the $|f|$-bit binary representation:

$$\|f\| = \sum_{j=1}^{|f|} f(j) 2^{|f|-j}. \tag{11}$$

The magnitude $|f|$ and the norm $\|f\|$ are both many-one functions from $\{0, 1\}^*$ onto the nonnegative integers, since there are $2^{|f|}$ strings of length $|f|$ and $\|0^k f\| = \|f\|$ for all $k \geq 0$. However, f is uniquely specified by the pair of integers $m = |f|$ and $k = \|f\|$, $m \geq 0$, $0 \leq k \leq M = 2^m - 1$. (We retain the use of $|S|$ as the number of members in the set S: the context will make clear whether the argument of $|\cdot|$ is a string or a set.)

If f is a concatenation $f = f_1 f_2 \cdots f_N$ of N shorter strings, then

$$|f| = \sum_{j=1}^{N} |f_j|, \quad f_j(k) = f\left(\sum_{i=1}^{j-1} |f_i| + k\right), \; k \in [1, |f_j|]. \tag{12}$$

For integer n and w, the set $\hat{T}(n, w)$ of N-entry, w-bit (canonical representations of)

tables is the set of binary-valued strings satisfying (12) with

$$N = 2^n - 1,\ W = 2^w - 1;\ |f_j| = w,\ \|f_j\| \in [0, W] \text{ for all } j \in [1, N];\ |f| = Nw. \quad (13)$$

(The notation $\hat{T}(n, w)$ is used to distinguish the set of bitstrings $f \in T$ from the corresponding set of N-tuples of integers $(\|f_1\|, \|f_2\|, \cdots, \|f_N\|)$ denoted by $T(N, W)$. Since canonical representation is a 1-1 map, $|\hat{T}(n, w)| = |T(N, W)|$.)

It takes Nw bits to store a string $f \in \hat{T}(n, w)$. No alternate 1-1 encoding of $\hat{T}(n, w)$ into $\{0, 1\}^L$ can take less, since $\hat{T}(n, w) = \{0, 1\}^{Nw}$ and there are not enough binary sequences to go around unless $L \geq Nw$.

Let $\hat{I}(n, w) \subseteq \hat{T}(n, w)$ be the subset of tables whose entries increase weakly in norm:

$$1 \leq j < k \leq N \rightarrow 0 \leq \|f_j\| < \|f_k\| \leq W = 2^w - 1. \quad (14)$$

An $f \in \hat{I}(n, w)$ is (the canonical representation of) an inventory, and $|\hat{I}(n, w)| = |I(N, W)|$. It takes the same Nw bits to store a string $f \in \hat{I}$ as to store any $f \in \hat{T}$. Since not all strings of length Nw are in $\hat{I}$ the representation is redundant. Fano [3] has given one which is more concise: we give a closely related one.

Let $w \geq n > 0$ and let b be an integer, $0 < b < w$. Define the b-bit *quotient* string q_j to be the first b bits of f_j, the remainder string r_j to be the last $w - b$ bits of f_j, and q and r as the resulting concatenations:

$$q_j(i) = f_j(i),\ i \in [1, b];\quad r_j(i) = f_j(b + i),\ i \in [1, w - b];\quad f_j = q_j r_j;$$
$$q = q_1 q_2 \cdots q_N \in \hat{I}(n, b),\ r = r_1 r_2 \cdots r_N \in \hat{T}(n, w - b). \quad (15)$$

Let u_j be the unary encoding of the difference $\|q_j\| - \|q_{j-1}\|$ into a string of 0's terminated by a 1, and let u be the concatenation of the resulting strings including a terminal string of 0's:

$$u_1 = 0^{\|q_1\|}1;\quad u_j = 0^{\|q_j\| - \|q_{j-1}\|}1,\ j \in [2, N];\quad u_{N+1} = 0^{B - \|q_N\|},\ B = 2^b - 1;$$
$$u = u_1 u_2 \cdots u_{N+1} = u(1)u(2) \cdots u(N + B), \quad (16)$$

since u contains just B 0's and N 1's.

The encoding (16) maps $\hat{I}(n, b)$ 1-1 onto binary sequences of uniform length $N + B$ containing just B 0's and N 1's. Since each 1 in any such sequence can be interpreted as ending a sequence of 0's (whose length may be zero), the inverse map is straightforward:

$$\|q_j\| = \text{number of 0's lying to the left of } j\text{th 1 in } u, \quad (17)$$

which follows from (16). Then the 1-1 map shows that

$$|\hat{I}(n, b)| = \binom{N + B}{N} = \binom{N + B}{B} = |\hat{I}(b, n)|, \quad (18)$$

and complementing the sequence u by setting

$$\bar{u}(j) = 1 - u(j),\ j \in [1, N + B];\quad \bar{u} = \bar{u}(1)\bar{u}(2) \cdots \bar{u}(N + B), \quad (19)$$

maps $\hat{I}(n, b)$ 1-1 onto $\hat{I}(b, n)$. The inventory which is decoded from the complement $\bar{u}$ of the unary encoding u of q is the complement $\bar{q}$ of q. Thus

$$\|\bar{q}_k\| = \text{number of 1's lying to left of } k\text{th 0 in } u. \quad (20)$$

Combining (19) and (20) gives the direct definition of the complement $\bar{q}$ of an inventory q without reference to u:

$$\|\bar{q}_k\| = |\{j \in [1, N] \mid \|q_j\| < k\}|, \qquad \|q_j\| = |\{k \in [1, B] \mid \|q_k\| < j\}|, \quad (21)$$

which is just the pair of definitions (3) and (4) rewritten for strings q_j, $\bar{q}_k$ rather than integers x_j, $\bar{x}_k$ by using the norm notation defined in (11). The relation between u, $\bar{u}$ and q, $\bar{q}$ is shown in Figure 1.

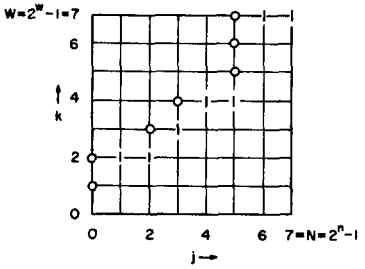

FIG. 1. The k-coordinates of points labeled by 1's are values of $q(j)$. There is one for each argument $j = 1, 2, \cdots, N = 2^n - 1$. The j-coordinates of points labeled with 0's are values of $\tilde{q}(k)$. There is one for each argument $k = 1, 2, \cdots, W = 2^w - 1$. Reading the labels from lower left to upper right gives the unary encoding $u = 00110101100011$. Complementing u gives $\bar{u}$, the unary encoding of $\tilde{q}$.

Given N and B, the string v consisting of the first $N + B - 1$ bits of u is also a 1-1 encoding of $q \in \hat{I}(n, b)$, since the value of the last bit of u is determined by the fact that u has B 0's and N 1's:

$$u(B + N) = N - \sum_{t=1}^{N+B-1} u(t) = N - \sum_{t=1}^{N+B-1} v(t). \tag{22}$$

Since either v or u determines q and by (15) q and r determine f, either of the concatenations vr, ur represent f 1-1 for any $b \in [0, w]$. It is easy to show that the storage required is minimized when $b = n$, so $B = N$: then $q \in I(n, n)$ is called a *square* inventory, and

$$\begin{aligned} |\, ur \,| &= |\, u \,| + |\, r \,| = 2N + N(w - n) = N(w - n + 2), \\ |\, vr \,| &= N(w - n + 2) - 1. \end{aligned} \tag{23}$$

We next show that no other 1-1 mapping of $\hat{I}(n, w)$ into binary sequences can do much better. Let $H(p, q)$ be the binary entropy function ($q + p = 1$, $0 < p < 1$):

$$H(p, q) = p \log (1/p) + q \log (1/q). \tag{24}$$

Wozencraft and Reiffen [9] show that

$$\begin{aligned} \log \binom{N + B}{N} &\geq (N + B)H\left(\frac{N}{N + B}, \frac{B}{N + B}\right) - \frac{3}{2} - \frac{1}{2} \log \frac{NB}{N + B} \\ &\geq \log \binom{N + B}{N} - \frac{1}{2} \log (4/\pi). \end{aligned} \tag{25}$$

Since $\log 4/\pi = 2 - \log \pi \doteq 0.348 < \frac{1}{2}$, (25) often determines the value of $\lceil \log \binom{N + B}{N} \rceil$ exactly. For example setting $B = N$ in (25) gives, from (13),

$$\begin{aligned} 2N - 1 - n/2 + \frac{1}{2} \log (4/\pi) + \frac{1}{2} (n - \log N) &\geq \log \binom{2N}{N} \\ &\geq 2N - 1 - \frac{1}{2} \log N > 2N - 1 - n/2. \end{aligned} \tag{26}$$

For $n \geq 3$, $n - \log N \leq \log \frac{8}{7}$ and $\frac{1}{2} \log (4/\pi) + \frac{1}{2} (n - \log N) < \frac{1}{2}$. Since for integer b and $\epsilon \leq \frac{1}{2}$,

$$b/2 + \epsilon \geq x > b/2 \rightarrow \lceil x \rceil = \lceil (b + 1)/2 \rceil, \tag{27}$$

checking the cases $n = 1$ and $n = 2$ proves part (i) of Theorem 2:

$$L_0 = \lceil \log |\, \hat{I}(n, n) \,| \rceil = \lceil 2N - (n + 1)/2 \rceil, \tag{28}$$

which compares to $|\, v \,| = 2N - 1$ from (23).

For general $\hat{I}(n, w)$ with $n \geq 1$, $s = w - n \geq 0$,

$$|\hat{I}(n, w)| = |I(N, W)| = \binom{N + W}{N} = \binom{2^n + 2^w - 2}{2^n}$$
$$= \binom{2^n + 2^w}{2^n} \cdot \frac{2^{n+w}}{(2^n + 2^w)(2^n + 2^w - 1)}$$
$$= \binom{2^w(1 + 2^{-s})}{2^n} \cdot \frac{2^{-s}}{(1 + 2^{-s})^2} \cdot \frac{2^n + 2^w}{2^n + 2^w - 1}.$$

Taking logarithms and using (25) gives

$$\begin{aligned}\log |\hat{I}(n, w)| &\geq 2^n(s + (1 + 2^s) \log (1 + 2^{-s})) - \tfrac{3}{2} - n/2 + \tfrac{1}{2} \log (1 + 2^{-s}) \\ &\quad - s - 2 \log (1 + 2^{-s}) + \log ((2^n + 2^w) / (2^n + 2^w - 1)) \\ &> (2^n - 1)(s + (1 + 2^s) \log (1 + 2^{-s})) \\ &\quad - \tfrac{3}{2} - n/2 + (2^s - \tfrac{1}{2}) \log (1 + 2^{-s}),\end{aligned} \tag{29}$$

where the second line drops the last term of the first. Using $\log (1 + x) \geq x$ for $0 \leq x \leq 1$, and the fact that $(1 + x) \log (1 + 1)/x)$ is monotone decreasing and so greater than $\log e$ which is its limit at $x = \infty$,

$$\begin{aligned}\log |\hat{I}(n, w) &\geq N(s + (1 + 2^s) \log (1 + 2^{-s})) - 1 - n/2 \\ &> N(s + \log e) - 1 - n/2,\end{aligned} \tag{30}$$

proving part (i) of Theorem 3. The error in the bottom line of (29) is less than 1 bit and the error in the top line of (30) is less than 2 bits for all $w \geq n \geq 1$.

4. *Retrieval Algorithms for Square Inventories*

Given $f \in \hat{I}(n, w)$, $w \geq n$, let $q \in \hat{I}(n, n)$ be its square n-bit quotient as in (15) and let u, the unary encoding of the differences of q as in (16), be stored in a memory $m = u$: $m(t) = u(t)$, $t \in [1, 2N]$.

The algorithm FIND Q $\varnothing$ finds q_j given $j \in [1, N]$ by accessing m and counting its 0's, 1's, and bits in three counters,

$$J = \sum_{t=1}^{T} m(t), \quad K = \sum_{t=1}^{T} (1 - m(t)), \quad T = J + K. \tag{31}$$

FIND Q $\varnothing$ needs a pair of initial values $J = J_0$, $K = K_0$ which satisfy (31), and works when the values are such (say $J_0 = K_0 = 0$) that $J_0 < j$.

FIND Q $\varnothing$

```
Enter
Set T = J + K
Set U = m(T + 1)
Add U to J
Add 1 − U to K
If J < j return to Enter
Exit.
```
(32)

When the count J in the 1's counter first reaches j, $m(T)$ is the jth 1 in u and ends the jth codeword u_j, so that the count in the 0's counter is

$$K = K(\min T \mid J = j) = \| q_1 \| + \sum_{i=2}^{j} (\| q_i \| - \| q_{i-1} \|) = \| q_j \|, \tag{33}$$

and the rightmost n bits of K are q_j. At that point the algorithm has accessed just

$$j - J_0 + \| q_j \| - K_0 \tag{34}$$

bits of u in m.

FIND Q 1 accesses the same $m = u$, but counts down rather than up, and works when its initial values (say $J_1 = K_1 = N$) satisfy (31) and are such that $J_1 \geq j - 1$.

FIND Q 1

Enter
If $J < j$, add 1 to J and go to Exit
Set $T = J + K$
Set $U = m(T)$ (35)
Subtract U from J
Subtract $1 - U$ from K
Return to Enter
Exit.

When it exits, FIND Q 1 has set $J = j$ and the n rightmost bits of K are again q_j. The number of bits accessed in m by FIND Q 1 is just

$$J_1 - (j - 1) + K_1 - \| q_j \|. \tag{36}$$

The complementary algorithms FIND $\bar{Q}$ $\varnothing$ and FIND $\bar{Q}$ 1 are obtained by replacing $m(T + 1)$ by $1 - m(T + 1)$ in FIND Q $\varnothing$ and $m(T)$ by $1 - m(T)$ in FIND $\bar{Q}$ 1. Given j and suitable starting values these algorithms exit with $J = j$ and $K = \| \bar{q}_j \|$, accessing m a number of times given by (34) and (36) with $\|q_j\|$ replaced by $\| \bar{q}_j \|$.

To answer the *identity* question about q (or $\bar{q}$), append to FIND Q $\varnothing$ (or FIND $\bar{Q}$ $\varnothing$) the commands

Print rightmost n bits of K
If $J = N$, halt
Add 1 to J (37)
Return to Enter.

The resulting archival algorithm FIND Q A (or FIND $\bar{Q}$ A), given $j = 1$ and $J_0 = K_0 = 0$, prints the N values of q_j (or $\bar{q}_j$), $j \in [1, N]$, in order, using the values $J = j$, $K = \| q_j \|$ (or $\| \bar{q}_j \|$) found on the Jth loop to start the $(J + 1)$-st, and accessing at most the $2N$ bits of u from m. For $n \geq 2$, $2N \leq nN$ and it is cheaper to print q by accessing u than by accessing q, both in bits stored and in bits accessed.

For storage and archival access u and its associated algorithms FIND Q A, FIND $\bar{Q}$ A comes very close to the lower bound L_0 for L, A_{da}, A_{ia} of Theorem 1. FIND Q and FIND $\bar{Q}$ don't do as well at finding single values of q_j or $\bar{q}_i$. Using $J_0 = K_0 = 0$ for FIND Q $\varnothing$ and $J_1 = K_1 = N$ for FIND Q 1 and averaging (34) and (36) gives an average of

$$A_d = A_i = N + \tfrac{1}{2} \tag{38}$$

bits accessed in m per direct (or inverse) question, independent of j and q, compared to the lower bound of n in Theorem 1.

To do much better than (38) requires the use of a directory to give better starting values for the algorithms. It is easy to enter the algorithms with a value q_i obtained from a directory because of a self-addressing property of u. Given $j \in [1, N]$, starting values $J_0 = i$, $K_0 = \| q_i \|$ will work for FIND Q $\varnothing$ if $i < j$, and $J_1 = i - 1$, $K_1 = \| q_i \|$ will work for FIND Q 1 if $i \geq j$.

5. *Directories*

Let a memory m^1 store the canonical representations of an inventory $q^1 \in I(n_1, n_1)$ and of its complement $\bar{q}^1 \in \hat{I}(n_1, n_1)$, and let table lookup algorithms SEEK Q (1) evaluate $q_j{}^1$ and SEEK $\bar{Q}$ (1) evaluate $\bar{q}_j{}^1$, each with an average of A^1 bit accesses to m^1, where

$$| m^1 | = | q^1 \bar{q}^1 | = 2n_1 N_1, \qquad A^1 = n_1. \tag{39}$$

Let $n_2 > n_1$ and define

$$s = n_2 - n_1, \qquad S = 2^s.$$

Given an inventory $q^2 \in \mathcal{I}(n_2, n_2)$, let the q^1 and $\tilde{q}^1$ above be the n_1-bit quotients of every Sth entry of q^2 and $\tilde{q}^2$, and let r^1 and $\tilde{r}^1$ be the corresponding s-bit remainders

$$q^2_{sj} = q_j{}^1 r_j{}^1, \qquad \tilde{q}^2_{sj} = \tilde{q}_j{}^1 \tilde{r}_j{}^1, \qquad j \in [1, N_1]. \tag{41}$$

Finally, let u^2 be the unary encoding of the differences of q^2. Then in the memory $m^2 = m^1 r^1 \tilde{r}^1 u^2$ of size

$$|\, m^2 \,| = |m^1\,| + |\, r^1 \,| + |\, \tilde{r}^1 \,| + |\, u^2 \,| = |\, m^1 \,| + 2N_1 s + 2N_2\,, \tag{42}$$

the prefix $\hat{m}^1 r^1 \tilde{r}^1$ is a directory to u^2 for both direct and inverse questions.

Given $j \in [1, N_2]$, an algorithm SEEK Q (2) $\varnothing$ evaluates $q_j{}^2$, in the following steps.

(i) If $\llcorner j/S \lrcorner = 0$, set $J_0 = K_0 = 0$ and go to FIND Q $\varnothing$, which exits with $K = \| q_j{}^2 \|$.

(ii) If $\llcorner j/S \lrcorner \neq 0$, call SEEK Q (1) which finds $q^1_{j/s}$. Then call a table lookup algorithm to find $r^1_{\llcorner j/s \lrcorner}$. Set

$$J = S_{\llcorner j/S \lrcorner}, \quad K = \| q^1_{\llcorner j/S \lrcorner} r^1_{\llcorner j/S \lrcorner} \| = \| q^2_{S\llcorner j/S \lrcorner} \|. \tag{43}$$

(iii) If $j = S_{\llcorner j/S \lrcorner}$, the answer was in the directory. Exit leaving $K = \| q_j{}^2 \|$.

(iv) If $j \neq S_{\llcorner j/S \lrcorner}$, go to FIND Q $\varnothing$ with starting values (43). FIND Q $\varnothing$ exits with $K = \| q_j{}^2 \|$.

SEEK Q (2) 1 is the descending counterpart to SEEK Q (2) $\varnothing$. If $\ulcorner j/S \urcorner = N_1$ it sets $J_1 = K_1 = N_2$ for FIND Q 1. If not, it calls SEEK Q (1) and table lookup to find $q^1_{\ulcorner j/s \urcorner}$ and $r^1_{\ulcorner j/s \urcorner}$ and sets

$$J = S\ulcorner j/S \urcorner - 1, \; K = \| q^1_{\ulcorner j/s \urcorner} r^1_{\ulcorner j/s \urcorner} \| = \| q^2_{s\ulcorner j/s \urcorner} \|. \tag{44}$$

If $j = S\ulcorner j/S \urcorner$, SEEK Q (2) 1 exits leaving $K = \| q_j{}^2 \|$. If not it calls FIND Q 1 with starting values (44), and FIND Q 1 exits with $K = \| q_j{}^2 \|$.

SEEK Q (2) accesses A^1 bits in m^1 and s bits in r^1 except when $\llcorner j/S \lrcorner = 0$ (or $\ulcorner j/S \urcorner = N_2$). This gives an average over $j \in [1, N_2]$ of

$$A^1(1 - (S-1)/N_2), \qquad s(1 - (S-1)/N_2) \tag{45}$$

accesses to m^1 and r^1 respectively for either the ascending or descending algorithm. It then calls FIND Q. Using the starting values (43) and (44) in (34) and (36), FIND Q finds $q_j{}^2$ with

$$\begin{aligned} &j - S_{\llcorner j/S \lrcorner} + \|q_j{}^2\| - \| q^2_{S\llcorner j/S \lrcorner} \| \text{ accesses to } u^2 \text{ for FIND } Q\ \varnothing, \\ &S\ulcorner j/S \urcorner - j + \| q^2_{S\ulcorner j/S \urcorner} \| - \| q_j{}^2 \| \text{ accesses to } u_j{}^2 \text{ for FIND } Q\ 1, \end{aligned} \tag{46}$$

where $\| q^2_{N_2+1} \|$ and $\| q_0{}^2 \|$ are interpreted as N_2 and 0 respectively.

Summing the two expressions in (46), summing further over $j \in [1, N_2]$, and dividing by $2N_2$ gives an average, over the two algorithms and the N_2 questions, of

$$\begin{aligned} (1/2N_2) \sum_{j=1}^{N_2} &(S(\ulcorner j/S \urcorner - \llcorner j/S \lrcorner) + \| q^2_{S\ulcorner j/S \urcorner} j/s \urcorner \| - \| q^2_{S\llcorner j/S \lrcorner} \| \\ &= (1/2N_2)((S-1)(N_2+1) + \| q^2_{N_2+1} \| - \| q_0{}^2 \|) \\ &= (S-1)(1 + 1/2N_2) \end{aligned} \tag{47}$$

bits accessed in u^2. Adding (45) and using $s = n_2 - n$, $A_1 = n_1$ gives an average of

$$\begin{aligned} A^2 &= (A^1 + s)(1 - (S-1)/N_2) + (S-1)(1 + 1/2N_2) \\ &= n_2 + S - 1 - (S-1)(2n_2 - 1)/2N_2 \\ &\leq n_2 + S - 1 = n_2(1 + S/n_2) - 1 \end{aligned} \tag{48}$$

bits accessed in m^2 by SEEK Q (2) in finding $q_j{}^2$. By the symmetry of m^1 and m^2, an algorithm SEEK $\tilde{Q}$ (2) calling FIND $\tilde{Q}$ can find $\tilde{q}^2{}_j$ with the same average number of accesses to m^2. At $S = N_2 + 1$ (i.e. $n_2 = n$, $n_1 = 0$, no directory), A^2 reduces to the value $N_2 + \frac{1}{2}$ of (38).

From (42) and (39) the required storage is

$$\begin{aligned} |\, m^2 \,| &= 2N_1n_1 + 2N_1S + 2N_2 \\ &= 2(N_2 + 1)(1 + n_2(N_1 + 1)/(N_2 + 1)) - 2(1 + n_2) \\ &= 2N_2(1 + n_2/S) - 2(n_2 - n_2/S) \end{aligned} \tag{49}$$

since $S = (N_2 + 1)/(N_1 + 1)$.

To prove Theorem 2 let $n_2 = n$, $\delta_n = n/S$. Then for square inventories $A_d = A_i = A^2 = n(1 + 1/\delta_n) - 1$, $L = |\, m^2 \,| = 2N(1 + \delta_n) - 2(n - \delta_n)$. Let $\Delta/2$ be the difference between $|\, m^2 \,|$ and its upper bound $(1 + \delta)L_0$ in Theorem 2. Using (28) for L_0,

$$\begin{aligned} \Delta &= 2((1 + \delta_n)L_0 - |\, m^2 \,|) \\ &= 2(1 + \delta_n)\lceil 2N - (n + 1)/2 \rceil - 4N(1 + \delta_n) + 4(n - \delta_n) \\ &\geq n(3 - \delta_n) - (1 + 5\delta_n). \end{aligned} \tag{50}$$

Choose the integer $s_n = \log S_n$ so that $1/2^{\frac{1}{2}} < \delta_n = n/S_n < 2^{\frac{1}{2}}$.

For the (n, δ_n) pairs $(3, \frac{3}{4})$, $(4, 1)$, $(5, \frac{5}{4})$ and $(n \geq 6, \delta < 2^{\frac{1}{2}})$, the last line of (50) shows that $\Delta > 0$. For $n = 2$, $\delta_2 = 1$ the second line shows $\Delta = 0$. For $n = 1$, choose $m = q\bar{q}$ giving $|\, m \,| = 2 = 2L_0$, $A_d = A_a = 1 = n = 2n - 1$, $A_{da} = A_{ia} = 1 = L_0$, proving (ii) in the theorem for $n = 1$. Setting $\epsilon_n = \max\{\delta_n, 1/\delta_n\}$ completes the proof of (ii) for all but the archival questions.

For the archival questions with $n \geq 2$, the algorithms FIND Q A, FIND Q A access u^2 in m^2 without using the directory, giving

$$\begin{aligned} (1 + n/2N)L_0 &\geq (1 + n/2N)(2N - (n + 1)/2) \\ &= 2N - \tfrac{1}{2} + (n/2)(1 - (n + 1)/2N) \\ &\geq 2N = A_{da} = A_{ia}. \end{aligned} \tag{51}$$

Only the limit $\epsilon_n \to 0$ remains to be proved. It needs more directories. Let

$$s_t = n_{t+1} - n_t, \quad S_t = 2^{s_t}, \quad m^{t+1} = m^t r^t \bar{r}^t u^{t+1} \tag{52}$$

and construct SEEK Q $(t + 1)$ like SEEK Q (2) except that it calls SEEK Q (t) to access the directory m^t, table lookup to access r^t, and FIND Q to access u^{t+1}.

The derivations of (48) and (49) permit starting with A^t and m^t for any t, not only $t = 1$. Using $m^t r^t r^t$ as a directory to m^{t+1} gives

$$\begin{aligned} A^{t+1} &= (A^t + s_t)(1 + (S_t - 1)/N_{t+1}) + (S_t - 1)(1 + 2/N_{t+1}) \\ &\leq A^t + s_t + 2^{s_t} - 1, \\ |\, m^{t+1} \,| &= |\, m^t \,| + 2N_t s_t + 2N_{t+1}. \end{aligned} \tag{53}$$

Two applications of (53), with $A^1 = n_1$, $|\, m^1 \,| = 2N_1 n$, and $s_1 = s_2 = s$, gives $n_3 = n_1 + 2s$, and

$$\begin{aligned} A^3 &= A^1 + s + 2^s - 1 + s + 2^s - 1 = n_3 + 2(2^s - 1), \\ |\, m^3 \,| &= 2N_3 + 2N_2(s + 1) + 2N_1(s + n_1) \\ &\leq 2N_3(1 + (s + 1)2^{-s} + (n_3 - s)2^{-2s}). \end{aligned} \tag{54}$$

Setting $s = \lceil (2/3) \log n_3 \rceil$ in (54) gives

$$\lim_{n_3 \to \infty} (A_3/n_3) = 1, \qquad \lim_{n_3 \to \infty} (|\, m^3 \,|/2N_3) = \lim_{n_3 \to \infty} (|\, m^3 \,|/L_0) = 1$$

and completes the proof of Theorem 2.

6. *Arbitrary Inventories*

To answer direct or inverse questions about an arbitrary inventory $f \in I(n, w)$, $w \geq n$, let

$$n_{t+1} = n, \qquad N_{t+1} = N = 2^n - 1, \tag{55}$$

let $q \in I(n, n)$ be the n-bit square quotient of f and $r \in \hat{T}(n, w-n)$ its $(w-n)$-bit remainder, with $f_j = q_j r_j$, $q = q^{t+1}$, $r = r^{t+1}$, and represent f in the memory m, where $m = m^{t+1}r^{t+1} = m^{t+1}r$, $|m| \leq 2N(1+\epsilon_n) + N(w-n) = N(w-n+2(1+\epsilon_n))$ by Theorem 2.

For direct questions, given $j \in [1, N]$ the algorithm SEEK F $(t+1)$ first calls SEEK Q $(t+1)$ which finds q_j^{t+1}, then reads the $w-n$ bits of r_j^{t+1} by table lookup in m, and finally prints the concatenation $f_j = q_j^{t+1}r_j^{t+1}$ with an average of

$$A_d \leq A^{t+1} + w - n \leq n(1+\epsilon_n) - 1 + w - n < w(1+\epsilon_n) \tag{56}$$

accesses to m.

For inverse questions, given $k \in [1, W]$ let j be its n-bit quotient and h its $(w-n)$-bit remainder:

$$j = \lfloor k/2^{w-n} \rfloor, \qquad h = k - 2^{w-n}j. \tag{57}$$

By the definition (3) of $\hat{f}$,

$$\begin{aligned} \|\hat{f}_k\| &= |\{i \in [1, N] | \, \|f_i\| < k\}| = |\{i \in [1, N] | \, 2^{w-n}\|q_i\| + \|r_i\| < k\}| \\ &= \|\bar{q}_j\| + |\{i \in [1, N] | \, \|q_i\| = j, \ \|r_i\| < h\}|. \end{aligned} \tag{58}$$

To find $\|\hat{f}_k\|$, the algorithm SEEK $\bar{F}$ $(t+1)$ first finds the pair of values $\|\bar{q}_j\|$, $\|\bar{q}_{j+1}\|$. If $\|\bar{q}_j\| = \|\bar{q}_{j+1}\|$ then $\|\hat{f}_k\| = \|\bar{q}_j\|$ and the job is done. If not, add i to $\|\bar{q}_j\|$ for each $i \in [\|\bar{q}_j\| + 1, \|\bar{q}_{j+1}\|]$ such that $\|r_i\| < h$. The resulting sum is $\|\hat{f}_k\|$.

To find the pair $\|\bar{q}_j\|$ and $\|\bar{q}_{j+1}\|$ takes only one more average access to u^{k+1} than to find $\|\bar{q}_j\|$ alone. Modify SEEK $\bar{Q}$ $(t+1)$ to store an extra value and use the ascending algorithm to find $\bar{q}_{j+1}$ when j is even and the descending algorithm to find $\bar{q}_j$ when j is odd. Summing the two expressions in (46), then summing over the odd $j \in [1, N]$ and finally dividing by the $N+1$ possible pairs gives an average of

$$S/2 + S/2(N/(N+1)) = S(1 - 1/2(N+1)) < S \tag{59}$$

bits accessed per pair. (59) is less than 1 more accesses than the number (47) needed to find a single value, and adds less than 1 to the value of A^2 given by (48): thus for a pair,

$$\text{accesses} \leq A^2 + 1 \leq n(1+\epsilon_n). \tag{60}$$

(Note that for a pair, at $S - 1 = N$, (59) like (48) reduces to $N + \frac{1}{2}$: it costs no more accesses to find a pair of adjacent entries than to find one entry, in the unary encoding without directories.)

The remaining accesses made by SEEK $\bar{F}$ $(t+1)$ to m are used to compare the binary representation of h bit by bit to the remainders r_i with i $[\|\bar{q}_j\| + 1, \|\bar{q}_{j+1}\|]$, reaching the decision $\|r_i\| < h$ or $\|r_i\| \geq h$ at the first mismatch or at the $(w-n)$-th bit if all bits match. To find the average accesses to r_i we keep i fixed and run SEEK $\bar{F}$ $(t+1)$ once for each of the $W+1$ values $k \in [0, W]$.

An h will have its pth bit compared to the pth bit of r_i if the first $n+p-1$ bits of k match the first $n+p-1$ bits of f_i. Such a match happens for a fraction $2^{-(n+p-1)}$ of the $W+1 = 2^w$ values of $k \in [0, W]$. Thus the total accesses to r_i for all k will be

$$2^w \sum_{i=1}^{w-n} 2^{-(n+p-1)} = 2^{w-n} \cdot 2(1 - 2^{-(w-n)}).$$

Summing further over the N remainders r_i, $i \in [1, N]$, gives the total accesses to all remainders for all $W+1$ values of k. Dividing by the number W of questions $k \in [1, W]$ which actually get asked gives an average bounded by

$$2(1-2^{-(w-n)})2^w(2^n - 1)/(2^w - 1) \leq 2(1-2^{-(w-n)}) \tag{61}$$

(since $w \geq n$). Adding to (61) the $n(1+\epsilon_n)$ or fewer accesses needed to find the pair

$\| \bar{q}_j \|$, $\| \bar{q}_{j+1} \|$ from (60) gives the bound $A_i \leq n(1 + \epsilon_n) + 2(1 - 2^{-(w-n)})$ of Theorem 3 on average bits accessed in m by SEEK $\bar{F}$ $(t + 1)$ to find $\hat{f}_k$.

The archival questions don't need the directory. Modifications of SEEK F $(t + 1)$ and SEEK $\bar{F}$ $(t + 1)$ can find successive values of q_j and $\bar{q}_j$ by using FIND Q A of (36) and (37) accessing u^{t+1}, and use each value or pair of values together with accesses to r to find f_j and $\hat{f}_k$, with a total of $| u^{t+1} | + | r | = N(w - n + 2)$ or fewer accesses as given in (23). This completes the proof of Theorem 3.

7. *Summary*

The complexity of any algorithm is an upper bound to the complexity of the computation it carries out. Lower bounds are harder to come by. Information theoretic techniques can provide lower bounds to bit counts. Relevant bit counts for problems of retrieval and computation are: (i) the number of bits which must be stored to store the representation of one of a set of possible data bases or arguments, (ii) the number of bits which must be accessed in finding one of a set of possible answers to a question about the object whose representation is stored, (iii) the number of bits of state information required by the machine or algorithm which accesses the representation to answer the question, (iv) the number of bits of read-only memory or program or wiring information needed to specify the machine or algorithm given the number of its states, and (v) the number of bits in the representation which must be changed to represent an update of the data base. In this paper we have dealt quantitatively only with (i) and (ii), keeping an eye on (iii) and (iv) and ignoring (v) by dealing with static files. Quantitative treatment of (iii) is not difficult. Quantitative treatment of (iv) seems to be very hard because of the difficulty of finding a suitably technology-invariant measure of the complexity of combinational logic. Quantitative treatment of (v) is under exploration (see Flower [10]).

Lower bounds may be used to rule out impossible schemes quickly. More important, they act as a spur to the invention of schemes whose performance approaches the bounds, like those above. Lower bounds to bits stored and to bits accessed can be computed for a variety of retrieval questions, and it is possible in a number of cases simultaneously to attain or approach both a storage bound and an access bound for a particular type of question. The example given here is unusual in that it permits several different types of questions to be answered efficiently at near minimal storage cost. One question for a more general theory is what data structures and sets of retrieval questions permit such an approximation to all lower bounds at once. Another is to find the trading relations in the more typical case when not all minima can simultaneously be attained.

Acknowledgments. I would like to express my appreciation to Marvin Minsky and Seymour Papert, whose discussion both in [7] and in person aroused my interest in this class of problems; to Terry Welch and Robert Gallager, whose critical reading of portions of the work has been most helpful; and to Cecil H. Green, whose gift to the Electrical Engineering Department at MIT of a chair for use to rotation by faculty members exploring new directions of research made concentrated effort on this and other related research topics possible.

REFERENCES

1. Cover, T. M. Enumerative source encoding. *IEEE Trans. IT-19* (Jan. 1973), 73–77.
2. Elias, P. On binary representations of monotone sequences. Proc. Sixth Princeton Conference on Information Sciences and Systems, March 1972, Dep. of Electrical Engineering, Princeton U., Princeton, N. J., 1972, pp. 54–57.
3. Fano, R.M. On the number of bits required to implement an associative memory. Memorandum 61, Computer Structures Group, Project MAC, MIT, Cambridge, Mass., n.d.
4. Fano, R. M. *Transmission of Information*, MIT Press, Cambridge, Mass., and Wiley, New York, 1961.
5. Gallager, R. G. *Information Theory and Reliable Communication*. Wiley, New York, 1968.

6. LEHMER, D. H. Teaching combinatorial tricks to a computer. Proceedings of Symposia in Applied Mathematics, Vol. X, Combinatorial Analysis, Amer. Math. Soc., Providence, R.I., 1960, Ch. 1, pp. 5-31.
7. MINSKY, M., AND PAPERT, S. *Perceptrons*. MIT Press, Cambridge, Mass., 1969, pp. 215-225.
8. SCHALKWIJK, J. P. M. An algorithm for source coding. *IEEE Trans. IT-18* (May 1972), 395-399.
9. WOZENCRAFT, J. M., AND REIFFEN, B. *Sequential Decoding*. MIT Press, Cambridge, Mass., 1961, pp. 71-73.
10. FLOWER, R. A. Computer updating of a data structure. Quart Progress Rep. 110, Res. Lab. of Electronics, MIT, Cambridge, Mass., July 1973.

RECEIVED OCTOBER 1972; REVISED MAY 1973