Trans. Inform. Theory, vol. IT-23, pp. 343–353, May 1977.
[4] H. H. Tan, "Tree encoding of discrete-time abstract-alphabet stationary block-ergodic sources with a fidelity criterion," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 671–681, Nov. 1976.
[5] R. M. Gray, D. L. Neuhoff, and J. K. Omura, "Process definition of distortion-rate functions and source coding theorems," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 524–532, Sept. 1975.
[6] T. Hashimoto, "A direct proof of the process definition of the distortion-rate function for stationary ergodic sources," *Inform. Contr.*, vol. 51, pp. 45–57, 1983.
[7] T. Berger, *Rate Distortion Theory*, Englewood Cliffs, NJ: Prentice-Hall, 1971.
[8] F. Jelinek, "Tree encoding of memoryless time-discrete sources with a fidelity criterion," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 584–590, Sept. 1969.
[9] C. R. Davis and M. E. Hellman, "On tree coding with a fidelity criterion," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 373–378, July 1975.
[10] J. F. Flanagan, M. R. Schroeder, B. S. Atal, R. E. Crochiere, N. S. Jayant, and J. M. Tribolet, "Speech coding," *IEEE Trans. Commun.*, vol. COM-27, pp. 710–737, Apr. 1979.
[11] J. B. Anderson and J. B. Bodie, "Tree encoding of speech," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 379–387, July 1975.
[12] R. G. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.
[13] J. K. Omura and A. Shōhara, "On convergence of symmetric sources," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 573–577, July 1973.

A Universal Data Compression System

JORMA RISSANEN

***Abstract*—A universal data compression algorithm is described which is capable of compressing long strings generated by a "finitely generated" source, with a near optimum per symbol length without prior knowledge of the source. This class of sources may be viewed as a generalization of Markov sources to random fields. Moreover, the algorithm does not require a working storage much larger than that needed to describe the source generating parameters.**

I. Introduction

THE first universal data compression algorithms were capable of encoding strings, generated by independent information sources, with asymptotically optimum mean per symbol length without *a priori* given source probabilities (Davisson [2], Lawrence [4], Lynch [5], Schalkwijk [10]). Clearly, such algorithms estimate either directly or indirectly the statistics with increasing accuracy while the string is being encoded. The same approach can be extended, at least in principle, to all stationary sources by means of gathering the statistics of longer and longer segments. However, in practice there is an obvious difficulty of exponentially growing number of items to be stored, and new ideas are needed to do the job in a practically meaningful manner.

The most powerful universal algorithm published to date is due to Ziv and Lempel [12]. Their elegant algorithm achieves asymptotically optimum compression for strings generated by any stationary ergodic source, and it does the job in many cases in a quite practicable manner. Although the authors emphasize the finite machine nature of their data compression system and the associated notion of compressibility, the real power of the algorithm is its convenient data gathering capability. In order to see this as well as the limitations of the approach, we reinterpret their algorithm in a natural statistical framework of the type discussed in Rissanen and Langdon, [8]. This is done in Section II.

The main results in this paper are in Sections III, IV, and V. After having demonstrated in Section III that Ziv and Lempel's universal algorithm does not compress well the important class of strings generated by stationary random fields which arise, for example, in image compression applications, we describe a more powerful data gathering algorithm. Instead of partitioning the string into its "relevant" parsed segments of increasing length and collecting their occurrence counts, as done by Ziv–Lempel algorithm, our algorithm gathers the "contexts" in which each symbol of the string occurs, together with the associated conditional occurrence counts. The contexts, as subsets of the past string, have varying size, and they are in general overlapping. Instead of collecting all the possible sequences as contexts, which requires too much space, only the "relevant" ones are gathered. To find these the algorithm incorporates as a design parameter a rule which ranks the past symbols, relative to each symbol in the string, according to their importance in influencing the value of the symbol. At this stage of development, this rule for efficient operation is to be selected by the designer based upon the general nature of the string, but any rule will work.

In Section IV we complete the construction of the universal model by describing how to select a unique context for each symbol of the string from among the possible

Manuscript received February 3, 1982; revised July 30, 1982.
The author is with IBM Research, 5600 Cottle Road, San Jose, CA 95193.

0018-9448/83/0900-0656$01.00

ones. As, generally speaking, the achieved compression improves with the size of the contexts, which in turn increases their number and the complexity of the model, we associate a cost with each context. This cost is balanced against the incremental gain in compression due to its addition to the context space, and the result is a universal model with a new degree of "intelligence": the algorithm will find asymptotically any stationary ergodic "finitely generated" source from its samples. This means that with a reasonable choice of the design parameter the complexity of the model does not exceed appreciably that of the source. In contrast, the complexity of the Ziv–Lempel system when applied to the same string would increase beyond any bounds. The class of finitely generated sources is described in Section V.

II. A Reinterpretation of Ziv–Lempel Algorithm

We start with a brief description of the universal data compression system of Ziv and Lempel, [12]. This discussion also serves as an introduction to the main topic in this paper. The heart of their system is a so-called "incremental parsing algorithm," which parses the source string into a collection of segments of gradually increasing length. The rule is simple: starting with the empty segment, each new segment added to the collection is one symbol longer than the longest match so far found. For example, the string 010100010 gets parsed as the collection $\langle 0, 1, 01, 00, 010\rangle$. When the parsed segments are retained in the same order as they are received, each segment can be encoded as the ordered pair (i, y), where the index i, written as a binary integer, gives the position of the longest earlier found matching segment in the collection, and y gives the last added symbol. For example, the code of the segment "010" is, conceptually, the pair $(3, 0)$. We refer for further details of how the decoder can read the position index from the code string, to Ziv and Lempel [12].

The code length of string s is given by

$$L_{ZL}(s) = \sum_{j=1}^{n(s)} \lceil \log j \rceil + n(s), \tag{1}$$

where $n(s)$ denotes the number of parsed segments in s, and $\lceil x \rceil$ denotes the least integer not smaller than x. Despite the relatively crude coding procedure used by Ziv and Lempel, their universal data compression algorithm is asymptotically optimum for infinite strings generated by a stationary ergodic source, in that the per symbol compression converges to the per symbol entropy of the source. Clearly, the complexity of the implementation, in terms of the number of stored items needed to generate the parsing trees, grows beyond all bounds.

It was shown by Ma [6] that exactly the same sequence of parsed segments can also be generated with an older algorithm due to Tunstall [11]. Here is how: start with the initial parsing tree consisting of two leaves, where a weight of two is placed at the root and a weight of one at both leaves. Use this tree to parse the first segment as the path

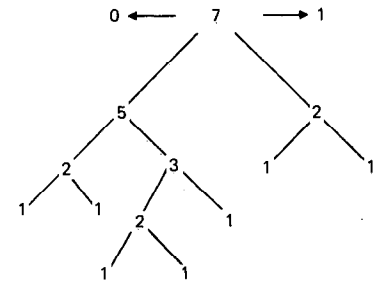

Fig. 1. Example of parsing algorithm.

from the root to a leaf. While climbing the tree, increment by one the count of each node visited. Hence the last leaf visited gets the count of 2, which is the maximum of all the leaf counts (because the others have the count of one). Split this leaf by creating two new nodes, and assign the count of one to both. This determines the new parsing tree, which is used to parse the next segment, and the cycle is repeated. As an example, we parse the string 010100010 with the same result as above. The final parsing tree is shown in Fig. 1, where the numbers at the nodes indicate the counts; we also drew the tree upside down.

We describe next how the parsing algorithm defines automatically a binary information source. As in [8], for this we need to generate a probability function, which assigns a probability $P(s)$ to every finite string such that the compatibility condition $P(s) = P(s0) + P(s1)$ holds. Here $s0$ and $s1$ denote the strings obtained by tagging 0 and 1, respectively, to the end of s. From the initial 2-leaf tree on, the above described algorithm defines a complete binary tree after each block is parsed. Let the (variable) string s have $n(s)$ parsed segments, and let $T(s)$ denote the binary tree defined by s. Each node's count equals the number of leaves that are its descendants, and, in particular, the root has the count $n(s) + 2$. Divide the node counts by the root count, and we have the machinery for defining the conditional probability of the next parsed segment as well as of any of its prefixes.

The algorithm defines the next segment to be one of the paths to the leaves in $T(s)$, while the intermediate nodes along this path correspond to the prefixes of this segment. If x denotes a prefix, set the conditional probability $P(x/s)$ as the ratio of the count of the node defined by x to the root count. In particular, when x is the entire parsed segment (i.e., when x defines a leaf) then $P(x/s) = 1/(n(s) + 2)$. Define the probability $P(sx)$ of the new string sx to be $P(s)P(x/s)$, and, because the probability of the empty string is 1, we get the probability of a string s with $n(s)$ full blocks as

$$P(s) = 1/(n(s) + 1)!. \tag{2}$$

The compatibility condition is immediately verified. As an example, suppose the string $s = 010001010$ with $T(s)$ given in Fig. 1 continues as 011. The conditional probabilities of all the symbols in the new segment are given by the tree as $P(0/s) = 5/7$, $P(1/s0) = 3/5$, and $P(1/s01) = 1/3$. These define the conditional probability $P(011/s) = 1/7$ for the sixth block 011.

The ideal code length for strings s having $n(s)$ full blocks is given by (2) as

$$-\log P(s) = \log (n(s) + 1)!. \tag{3}$$

This suggests an interpretation of the statistical model we have defined: a string s with $n(s)$ blocks is one of the $(n(s) + 1)!$ strings possible obtained by picking each of their blocks for $k = 1, 2, \cdots, n(s)$ randomly as one of the $k + 1$ leaves available at that point. Hence, with this type of model, the mean code length cannot be smaller than $-\log P(s) - \alpha((1 + n(s))!)$, where $\alpha(n) < \log \log n$ (Rissanen [7]), no matter how the coding is done. This clearly justifies the name "ideal code length." The length (3) is about one bit per block less than the length (1) in Ziv and Lempel's code.

The coding of s can be done practically by arithmetic coding, [8], in such a manner that the ideal code length is achieved for any string longer than, say, 1000 within 1 percent. In fact, even with the fast algorithm described in [3] the code length would exceed the ideal by not more than 2 or 3 percent.

III. A Context Gathering Algorithm

A major problem with the incremental parsing algorithm, and with all block models for that matter, is that it can capture random dependencies only between symbols that fall within one and the same block. Hence, for the large class of strings, where symbols interact in two or higher dimensional neighborhoods, any technique producing one-dimensional parsed segments is necessarily ineffective and often impossible to implement in a practicable manner. Consider, for example, a black and white printed document, which is partitioned into a fine grid of squares. By a simple thresholding device, each square may be regarded either as completely black or as completely white, and we may model the entire grid of squares as a sample of a two-dimensional field of binary random variables $x(i, j)$, where i and j are the coordinates specifying the position of a square. It is immediately clear that in a good model we should not regard the random variables $x(i, j)$ as independent, but rather we should try to fit a model which allows the color of the variable $x(i, j)$ to depend on the color of the neighboring variables. And there is clearly no reason to prefer the horizontal neighbors over the vertical ones.

In an attempt to capture such two-dimensional dependencies with parsed segments we may try to consider two-dimensional segments, i.e., rectangles. If we wish to apply the incremental parsing algorithm to collect the rectangles, we must define the notion of minimal extension and do the extension while maintaining the property that the created rectangles still partition the entire surface. A little thought will reveal that this cannot be done in an entirely satisfactory way. We could, for instance, consider rectangles of, say, k lines high and grow them to the right, but then we do not capture dependencies that are further up than k positions, except when the length of these rectangles is made wider than the width of the document. This we can in effect do by a "wrap around" procedure: when the k lines reach the right end of the document, the rectangle is continued along the next k lines from the left end. Now, to grow such huge rectangles cannot be done even in the simplest case where $k = 1$, which amounts to a linear representation of the grid. Indeed, if a document has the normal width of about 2000 symbols, then the algorithm would have to grow the binary tree to the depth of 2000 before any dependency of a symbol on the one lying immediately above it would start to show up, and the total number of symbols in the document would run out far before such a depth is reached. If we, again, try to avoid this difficulty by picking k so large, say 10, that we capture all the significant vertical dependencies, we run into another problem: the trees we must grow now have 2^{10} symbols. Again we run out of the string before any horizontal dependencies even between two adjacent symbols can be detected. These are the reasons why the Ziv–Lempel algorithm, despite its asymptotic optimality, is not able to reach even the vicinity of the ultimate compression within the length of any sample string.

In order to obtain more powerful models we must abandon the requirement that the collected segments partition the string. Instead, we plan to collect overlapping sets of symbols, each set defining a "context," on which symbol occurrences can be conditioned. As proved in [8], such conditioning allows for a more efficient way to take advantage of statistical regularities. In fact, even the segments found by the incremental parsing algorithm are contexts of each of their symbols, which interpretation offers a uniform and more fundamental explanation to the modeling efficiency of block models.

Formally, a context is defined by means of a total recursive function f: $B^* \to N$, where B^* denotes the set of all finite binary strings and N denotes the set of the natural numbers. The context $z(t)$ of the symbol $x(t)$, immediately following the "past" string $s = x(1) \cdots x(t-1)$, is the class of strings $s' = x'(1) \cdots x'(t-1)$ such that $f(s') = f(s)$. What this abstract definition really says is that the context of $x(t)$ is some (computable) function of the past string. That the context is an equivalence class means only that not every conceivable value for the past sequence of random variables need define a distinct context; some relevant feature of the past, common to several of them, is enough. Clearly, a measure of the degree of relevance is the context's capability of "skewing" the symbol's occurrence counts to lower its conditional entropy. As a simple example, consider the function f that maps any sequence (of length at least two) to the value defined by its last two symbols. Hence, the context of $x(t)$ is the binary number $x(t-2)x(t-1)$, as in the second-order Markov process.

We shall eventually describe a way to obtain the all-important structure function f, as it was called in [8], but in this section we give an algorithm that defines it partially. We consider for simplicity binary strings only, which we write as $s = x(1)x(2) \cdots x(t) \cdots$; we also let the same symbols $x(i)$ denote binary-valued random variables. The first step is to establish a ranking order for the past symbol positions relative to each symbol $x(t)$. The idea is to pick

that position first in which we judge the random variable to have the greatest influence on the value of the variable $x(t)$; then comes the next most influential position, and so on. Often, the geometric distance is taken as the basis for this; for example, in the string resulting from a linear scan of the document above with width M, we could pick the order to start as follows: first comes the preceding variable, $x(t-1)$, then the one right above, or $x(t-M)$ in the linear representation, next the left upper corner position, or $x(t-M-1)$.

To generalize, let $i \to t_i$ be a permutation of the natural numbers and define for any string $s = x(1) \cdots x(t-1)$ another $\sigma(s) = x(t-t_1) \cdots x(t-t_{t-1})$. In this, we have to decide how to set the values of symbols $x(t-t_i)$ whose index $t-t_i$ is nonpositive. This choice will have an effect only for small values of t, because we will be needing only about $\log t$ first members in the sequence $\sigma(s)$. The most natural choice is to define $\sigma(s)$ to consist of those consecutive symbols $x(t-t_1)x(t-t_2)\cdots$, only, whose indices are positive and, hence, which are symbols in s; if already the first symbol $t-t_1$ is nonpositive, define $\sigma(s)$ to be the empty string λ. This convention would be used in more practicable versions of the algorithm. However, if we set the values of the symbols with nonpositive indices to some arbitrary value, say 0, all the symbols of $\sigma(s)$ get defined, and this simplifies matters somewhat. For this reason, we adopt the latter convention. We write $\sigma(s)$ as the sequence $z_1 \cdots z_{t-1}$, where, accordingly, $z_i = x(t-t_i)$. In fact, the description of our algorithm does not involve the details of any particular permutation, and the reader may take it to be the identity permutation $t_i = i$, which means that $\sigma(s)$ is just the sequence s written in reverse: $x(t-1)\cdots x(1)$. We mention that in some applications there is a need for an even more general way to sort the past sequence, one that allows the sorting to change with t, but for the purposes of this paper the above described *sorting function* σ is adequate.

The idea in the next step is to grow two binary trees, one for the case where the current symbol $x(t)$, which we denote by u, has the value 0, and the other when it has the value 1. We are interested in the intersection of the two trees, which we actually generate directly; this is how:

1) We declare the context tree of the first symbol $x(1)$ in the string to be the 1-leaf tree $T(0)$, where the only node, the root, is marked with the pair of counts $(c(0,\lambda), c(1,\lambda)) = (1,1)$.
2) Proceeding recursively, let $T(t-1)$ be the last constructed tree with $(c(0,z), c(1,z))$ denoting the pair of the counts at node z. After the next symbol $u = x(t)$ is observed, generate the next tree $T(t)$ as follows: climb the tree $T(t-1)$, starting at the root and taking the branch, left for 0 and right for 1, indicated by each of the successive symbols in the past sequence $\sigma(x(1)\cdots x(t-1)) = z_1 z_2 \cdots$. For each node z visited, increment the component count $c(u,z)$ by one. Continue until a node w is reached whose count $c(u,w) = 1$ before the update.
3) If w is an internal node with node $w0$ as the left and $w1$ as the right successor, increment the component counts $c(u,w0)$ and $c(u,w1)$ by one. Define the resulting tree to be $T(t)$. If, again, w is a leaf, extend the tree by creating two new leaves $w0$ and $w1$. Assign to both leaves the same counts: $c(u,w0) = c(u,w1) = 1$ and $c(u',w0) = c(u',w1) = 0$, where u' denotes the opposite symbol to u. Call the resulting tree $T(t)$.

This completes the description of the algorithm, which we for future reference call "context."

As an illustration, we consider the binary string 10001, in which the past symbols are ordered by their distance from the current symbol, i.e., we use the identity permutation. After the first symbol 1 is observed, the second tree $T(1)$ is the 2-leaf tree with the marking $(1,2)$ at the root and the marking $(0,1)$ at both leaves. The second symbol 0 does not add any new leaves, by the first "if" clause in Step 3, but the root marking changes to $(2,2)$ and the leaf markings to $(1,1)$. The tree $T(3)$, generated by the third symbol 0, is given in Fig. 2(a), and the tree $T(5)$ after the last symbol 1 is given in Fig. 2(b).

Let z denote a node, which is defined by and identified with a past sequence $z_1 \cdots z_n$ as the path from the root to that node. Then the sibling nodes $z0$ and $z1$ denote the past sequences $z_1 \cdots z_n 0$ and $z_1 \cdots z_n 1$, respectively, while with $0, z$ and $1, z$ we denote the collections of events consisting of the "current" symbol $u = x(t)$ and the past sequence z. If we denote by $c(z)$ the sum of the counts at node z, then we may say that out of $c(z)$ occurrences of z the event u, z occurs $c(u,z)$ times, and we may define the conditional probability of the symbol u in context z as

$$P(u/z) = c(u,z)/c(z), \tag{3a}$$

provided $0 < P(u/z) < 1$. For example, in the tree $T(5)$ of Fig. 2, we have $P(x(t) = 0/z = 0) = 3/5$. Clearly, if a symbol, say $u = 0$, has not been observed at all in a context z, it does not mean that it will not occur in the future, and for coding purposes we must not assign the probability 0 to it. For this reason we put

$$P(u/z) = 1/(c(z)+1), \qquad \text{if } c(u,z) = 0. \tag{3b}$$

The way we eventually will select the context for each symbol makes the assignment (3b) possible only in the early part of the string, while the normal case is (3a). These conditional probabilities define the binary entropy

$$H(U;z) = -p\log p - (1-p)\log(1-p), \qquad p = P(0/z), \tag{4}$$

where z stands for a sequence $z_1 \cdots z_k$.

It follows from the updating and the node splitting rules in the algorithm that

$$c(u,z0) + c(u,z1) = c(u,z) \tag{5}$$

holds whenever the counts are greater than 1. By summing up over the two values of u, we get the compatibility

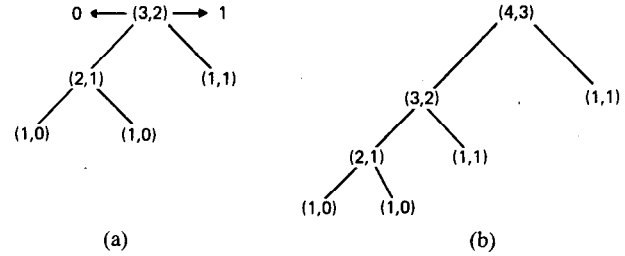

Fig. 2. Example of algorithm Context. (a) $T(3)$. (b) $T(5)$.

equation

$$c(z0) + c(z1) = c(z), \tag{6}$$

which holds if all the counts are greater than one. It is these two last equations that would not hold (except, approximately for long strings) if we had adopted the other rule for defining $\sigma(s)$, mentioned above, namely, one where the variables with nonpositive indices are dropped. These equations are by no means necessary for a proper behavior of the algorithm, but they do shorten the proof of the theorem in Section V.

Further, let Z denote the set of leaves defining a complete subtree, where for all the leaves $c(u, z) > 1$. Then by (6) the root count $c(\lambda)$ equals the sum of the leaf counts $c(z)$, and we may regard Z as a context-valued random variable with probability distribution given by

$$P(z) = c(z)/c(\lambda). \tag{7}$$

It can be seen from the algorithm that the node count $c(u, z)$ differs from the number of times symbol u occurs in context z in the so far processed string by at most a number that depends on z but not on the length of the string. This is because after the context z has been added to the tree as a node and its counts exceed 1, every occurrence of u in context z will be noted and the count incremented. Accordingly, only a few earlier occurrences of u in context z may have been omitted. As a result, the tree generated by the algorithm accumulates essentially all the relevant contexts and the associated symbol statistics as the length of the string grows.

IV. Universal Model

The trees $T(t)$, generated by the algorithm context, incorporate a stockpile of contexts for the symbols in the string, and as noted at the end of previous section, essentially all the past contexts are included. Specifically, the contexts for the symbol $x(t)$ correspond to the nodes met as we climb the tree $T(t-1)$ in Step 2 of the algorithm according to the past sequence $z_1 z_2 \cdots$. The question is which node we should pick as the context for $x(t)$. Suppose for the moment that we already have made the selection, i.e., we have associated with the symbol $x(t)$ a context $z^*(t)$ as some prefix of $\sigma(x(1) \cdots x(t-1))$. This really means that we then have defined a binary source $\langle B^*, P \rangle$, where $P(s)$ is defined as the product of the conditional probabilities of its symbols, (3a) or (3b). These probabilities can be used to encode each symbol, and the optimum code length for string $s = x(1) \cdots x(t)$ is given by

$$-\log P(s) = -\sum_{i=1}^{t} \log P(x(i)/z^*(i)). \tag{8}$$

The formula (8) suggests an intuitively attractive context selection rule: define $z^*(t)$ as the first node z in $T(t-1)$ along the path $z_1 z_2 \cdots$, where $P(x(i)/z)$ deviates most from $1/2$. Indeed, if in a stationary source each such context z^* would occur increasingly often with relative frequency $P(z^*)$, then the per symbol length from (8) would approach a conditional entropy of the type

$$\sum P(z) H(U; z) = H(U/Z), \tag{9}$$

where z runs through the set of the contexts. Moreover, because the rule minimizes the binary entropies $H(U; z)$ in (4), the entire conditional entropy and the per symbol mean length are clearly minimized. The main problem with this context selection rule is that the number of contexts $z^*(t)$ tends to increase with t even when the string is generated by an independent information source. The reason for this will be apparent later in the proof of Theorem 1, but intuitively it follows from the trend that conditioning on a larger set tends to lower the conditional entropy.

Instead of putting an arbitrarily selected bound to the number of contexts, we would like the algorithm itself to set the bound as it processes the string. The idea behind such an "intelligent" algorithm, which we adapt from the recent approaches to estimation theory, Akaike [1] and Rissanen [9], is to associate a cost with each context, and accept a context in the set Z only if its share in reducing the conditional entropy exceeds its cost. For this we need to calculate the increase in the conditional entropy (9) that results when two of the elements $z0$ and $z1$ in Z are fused into the parent node z. Call the reduced set Z'. The increase, which is independent of the set Z, is given by

$$\begin{aligned}\Delta(t, z) &= H(U/Z') - H(U/Z) \\ &= P(z) H(U; z) - P(z0) H(U; z0) \\ &\quad - P(z1) H(U; z1),\end{aligned} \tag{10}$$

where the probabilities and the binary entropies are defined by the node counts in $T(t-1)$, (3a), (3b), (4), and (7). The difference $\Delta(t, z)$ is also nonnegative by a well-known inequality; we shall later give a formula showing that.

We now describe a context selection rule as follows: define the context $z^*(t)$ for the symbol $x(t)$ to be the node in $T(t-1)$ with greatest length $|z|$, i.e., depth, along the path defined by the past sequence $z_1 z_2 \cdots$, such that

$$\Delta(t, z) > (1/t) \log t,$$

while

$$|z| \leqslant \beta \log t,$$

and

$$\min\{c(z0), c(z1)\} \geqslant 2\alpha t/\sqrt{\log t}, \tag{11}$$

or, if no such node exists, take z to be the root node. Here, α and β are positive numbers. Their purpose is to define the range of the nodes to be searched to be appropriate for a finite string, but for infinite strings any values for them will do. The upper bounds for z are selected for the purpose of making the proof of the main theorem easy. The first bound $\beta \log t$ is in most cases satisfied automatically, because the depth of the entire tree $T(t)$ is of the order of $\log t$. The purpose of the second bound is to make sure that the counts grow at the selected contexts, while not restricting the set of the contexts to be uniformly bounded. Observe that the count $c(z)$ for a string generated by a stationary source grows as $P^0(z)t$, where the source probability $P^0(z)$ tends to zero as the length of z grows. This is why the multiplier of t in the second bound is taken as a function that tends to zero as t grows.

We could mark in some manner the special context nodes in the tree, for example, by turning a "flag" bit to one, while keeping it at zero if a node is not a context. Then a creation of a new context turns an earlier found context node on its path to a nonflagged regular node, and since every path has a context, the set $Z(t)$ of them all in the tree $T(t-1)$ defines a complete subtree. Once the count ratios at the nodes are stabilized and if $Z(t) = Z$ remains unchanged, then as easily verified, this selection rule will minimize locally the combined cost

$$H(U/Z) + (|Z| \log t)/t, \tag{12}$$

where $|Z|$ denotes the number of elements in Z. By a local minimum we mean a set of leaves such that a fusion of any two leaves or a split of any leaf into two new leaves will increase the value of the sum in (12).

Why is the cost associated with each context taken as $(\log t)/t$? A pragmatic answer will be given in the theorem below, namely, that with such a cost we get the desired behavior. But we can also give a natural justification for the cost term. This is particularly easy if we instead of adaptation would first construct the context tree for the entire string and then using the so-found data would determine in the second pass the contexts of the symbols and calculate the ideal code length (8). Indeed, as each count is proportional to t, it takes about $\log t$ bits to write down each, and therefore about $|Z| \log t$ bits are needed to describe all the parameters defining the model. Because in such a two-pass data compression algorithm the model clearly must also be sent to the decoder along with the code string, the total code length is approximately as given in (12).

But it is a quite remarkable fact, which we were able to show only after this paper went to print, that the same cost term is valid even when the parameters are determined adaptively from the past string, as is done here. The intuitive reason for this is that the cumulative effect of the inherent estimation errors increases the ideal per symbol code length by $(\log t)/2t$ per parameter. Accordingly, the per symbol entropy (9) ought to be incremented by this much in order to account for the fact that we are using an estimate for the next symbol's probability, and a good context selection rule should minimize the total per symbol ideal code length, which is what (11) seeks to do. (The second term in (11) should really be divided by two, but this correction does not change the main theorem, Section V.)

V. Main Theorem

For the main theorem in this section we need to define a class of "finitely generated" sources, appropriate for random fields. Although these sources are technically Markov sources, we must describe them more efficiently by taking advantage of their random field nature. Just as in Section II, viewing them as Markov chains may involve an immense number of states with their transition probabilities, while viewing them as random fields allows us to describe a small number of conditional probabilities as generators, which with a simple extension rule define the rest. This simplification in the description of such sources is of crucial importance, if one wishes to construct an algorithm for their estimation.

Let $i_1 < \cdots < i_n$ denote n natural numbers, and let f^0 denote a function which maps every string $s = x(1) \cdots x(t)$, to its context

$$f^0(s) = x(t - i_1) \cdots x(t - i_n), \tag{13}$$

where we set $x(t - i_j)$ empty if $t - i_j \leqslant 0$. The range of f^0 clearly consists of the nodes of the balanced tree of depth n, called the (*source*) *contexts*. We denote by Z^0 the set of the 2^n leaves, defined by the values of the variables $x(t - i_1), \cdots, x(t - i_n)$. We call a stationary ergodic process *finitely generated*, if for all finite strings su,

$$P^0(su) = P^0(s)P^0(u/f^0(s)). \tag{14}$$

Observe that with this rule the random variables $x(t), x(t') \cdots$, conditioned on the same context z at different locations t, are independent.

The conditional probabilities $P^0(0/f^0(s))$ and $1 - P^0(0/f^0(s))$ cannot be chosen entirely freely to generate a stationary process, for they must also satisfy the conditions for stationarity, namely, $P(0s) + P(1s) = P(s)$ for all s (this nonorthodox but handy way of defining stationarity was done in [8]). However, it is clear that these conditions can be satisfied, and we may regard the class of finitely generated processes as being well defined. They clearly include all Markov processes with fixed transition probabilities. Finally, we also assume that the generating index set $i_1, \cdots, i_n$ is unique (so that it can be estimated), which in particular implies that it is minimal, i.e., that none of its proper subsets defines the same probabilities (13).

The per symbol entropy of a stationary finitely generated source is seen to be given by the conditional entropy

$$H^0(U/Z^0) = \sum_{z \in Z^0} P^0(z)H^0(U; z), \tag{15}$$

where $H^0(U; z) = -p \log p - (1 - p)\log(1 - p)$ with $p = P^0(0/z)$.

We define one more notion for the theorem to follow. For a sorting function $\sigma(s) = x(t - t_1)x(t - t_2) \cdots =$

$z_1 z_2 \cdots$ denote the set of the shortest past sequences $z_1 \cdots z_m$ that contain all the random variables $x(t - i_1), \cdots, x(t - i_n)$ defining the source by Z_σ. Hence, m is the index for which $t_m = i_n$. An example showing the relationship between the two sequences is given in the discussion following the theorem.

Theorem 1: Let s be an infinite string from a finitely generated stationary ergodic source defined by the indices $i_1, \cdots, i_n$, and let the algorithm Context generate the tree $T(t-1)$ for the growing prefix s^t of s, using any sorting function σ. Then with the context selection rule (11), for almost all samples s,

$$\Pr\{z^*(t) \in Z_\sigma\} \to 1, \qquad \text{as } t \to \infty. \tag{16}$$

Moreover,

$$-(1/t) \cdot \log P(s^t) \to H^0(U/Z^0), \qquad \text{as } t \to \infty. \tag{17}$$

We give the proof in the Appendix.

The following corollary is worth mentioning.

Corollary: Let the source be a Markov source of some (unknown) order n. Then (16) and (17) hold, where $Z_\sigma = Z^0$ is the set of the 2^n states.

VI. Discussion

By Theorem 1 the universal model is capable of containing the source generating contexts within a finite set Z_σ, whose size depends on the sorting function σ. This means that we do not need to grow the trees $T(t)$ much beyond Z_σ. In practice, we can let the trees $T(t)$ grow until they have, say, twice the number of leaves in the maximum set of contexts z^* found. Because the maximum context set depends on $T(t)$, and we wish to maintain $T(t)$ twice the size of the maximum context set, there is a bit of "dog-chasing-its-tail" process involved. This is easily resolved, however. We can, for example, partition the indefinitely long string into a set of growing segments of length $K, 2K, 3K, \cdots$, where K is some positive integer. During the first segment we let the tree $T(t)$ grow freely, but in the subsequent intervals we let it grow only until it has twice the number of nodes in the context set found in the previous interval. It is easily seen that this process lets the context set grow to its final size and shape, as stated in the theorem, and the trees $T(t)$ stay bounded too.

Although the theorem holds for any function σ we select, the size of the final context tree and, therefore, the size of the maximum of the trees $T(t)$ and the implementation cost of the algorithm are strongly dependent on the selection. The best case, of course, is when we know the ranking of the past symbols and hence the sorting function. This is still a nontrivial modeling problem, because we must locate a subset Z^0 that generates all the contexts of the source, among the set of all possible contexts, namely, the set B^* of all binary strings. In this case the algorithm will generate the trees $T(t)$ until they have about twice the number of leaves in Z^0, no matter how long the string is. In contrast, for example, the Ziv–Lempel algorithm would keep on generating longer and longer parsed blocks until it runs out of work space, without ever realizing that no further compression is being gained.

In many cases we do not, of course, know the sorting function σ. However, this universal algorithm provides a means of experimentally estimating it, which we regard as the single most important and unique feature of the algorithm, not shared by any other known to us. We can fix the maximum permitted size for the trees $T(t)$ and run the algorithm for several different sorting functions. The one giving the best compression evidently captures the most efficient contexts. We believe that such exploratory study can give valuable information of the strings to be compressed, and make the crucial modeling problem a little more mechanical. Bear in mind that the problem of finding the best model is undecidable, and all we can do is try to build more and more intelligent algorithms to aid us in finding good models if not the very best.

We illustrate with a simple example the effect of choosing a good and a bad sorting function to the size of the needed workspace. Suppose the source is generated by the four conditional probabilities $P(0/x(t-1)x(t-100))$, corresponding to the four values of the two indicated variables. In a two-dimensional field of 100 symbols wide, this means that a symbol's value is statistically dependent only on the previous value and the one right above it. If we choose for the model the sorting function σ defined by the identity permutation, then Z_σ consists of the 2^{100} sequences of length 100, and the algorithm will not find the second generating context within any string of practical size. We have the same problem as with the Ziv–Lempel algorithm. But if we guess that the nearby symbols are the significant symbols, and we put $\sigma(s) = x(t-1)x(t-2)x(t-99)$ $x(t-100) \cdots$, then Z_σ is defined by the 16 sequences of length 4. This time, the algorithm finds both of the generating contexts quickly with a small workspace. Observe, that it is not necessary to guess the precise ranking of the symbols in order to achieve asymptotically the ideal compression with a limited workspace.

What about compressing a string which cannot be well modeled by a finitely generated source? Again, by guessing a reasonable sorting function we can let the algorithm generate the trees $T(t)$ and the contexts, but this time the trees grow until a preset bound for the workspace is reached. The situation is quite analogous to the Ziv–Lempel algorithm, and the only advantage is that for the same amount of workspace we get a better compression if a sorting function sufficiently different from the one defined by the identity permutation is required. In the case with the identity permutation, the roles are reversed, because then the best contexts are to be found among the immediately preceding symbols, which the Ziv–Lempel algorithm is capable of taking advantage of. This is accomplished with a somewhat smaller workspace. The number of nodes in the tree built by the incremental parsing algorithm for a string of length t is approximately $t/\log t$, while the tree $T(t)$ built by the algorithm context has t

nodes. In typical cases the ratio of the workspace size in favor of Ziv–Lempel algorithm is then something like 10–20. The final compressions in both cases are about the same, because one can simulate, as it were, the block model created by the incremental parsing algorithm as a binary model using contexts.

We conclude this section with two more remarks. In order to obtain a universal data compression system with per symbol code length near the ideal (16), all we need to do is to apply an arithmetic binary code to encode each symbol $u = x(t)$ using the conditional probability $P(u/z^*(t))$. We have written a computer program for the algorithm context in a slightly modified form in the general case with any finite number of symbols. An implementation of the minimum conditional entropy rule, mentioned in the beginning of Section V, is straightforward, although a new problem not present in the binary case arises, namely the problem of how to assign a distribution for an alphabet of which only a small number of symbols has yet occurred. Several reasonable solutions exist, and with one of them, strings defined by English text can be compressed very well with a reasonable workspace size. Notice that the storage space explodes if we model such strings by Markov sources of some order $k > 1$.

Appendix

Proof of Theorem 1: The first task is to evaluate $\Delta(t, z)$ at a node z of $T(t-1)$ in terms of the probabilities (3a), (3b), (4), and (7), defined by the counts. The result is, as can be verified by a direct computation, the familiar formula

$$\Delta(t,z) = P(z) \sum_{y=0,1} P(y/z) \sum_{u=0,1} P(u/zy) \cdot \log [P(u/zy)/P(u/z)], \quad \text{(A1)}$$

where for simplicity we did not indicate that the probabilities depend on t. Also, we use the same symbol P for all the conditional probabilities and let the argument variables tell which one is meant; for instance, $P(u/z)$ is not the same function as $P(y/z)$. Moreover, by a well-known result, $\Delta(t/z) \geq 0$, and it is zero if and only if for all u, z, and y, $P(u/zy) = P(u/z)$.

We show first that any node z of Z_σ is a prefix of some context $z^*(t)$ for large enough t. We have nothing to prove if the source is independent, because then Z_σ consists clearly of the root λ, only. Otherwise, let z be an interior node with depth $m-1$ of T_σ, the tree defined by Z_σ as the set of leaves. Then its descendants zy are leaves in Z_σ, and they contain a source generating context, i.e., a sequence of the values $x(t-i_1) \cdots x(t-i_n)$, as a subsequence. By the minimality of the source generating contexts, the source probability $P^0(u/zy)$ cannot equal $P^0(u/z)$ for both values of u, and

$$\Delta^0(z) = P^0(z) H^0(U; z) - P^0(z0) H^0(U; z0) - P^0(z1) H^0(U; z1) > 0.$$

By the ergodic theorem, $\Delta(t, z) \to \Delta^0(z)$, and we conclude with (11) that z is a proper prefix and zy a prefix of $z^*(t)$ for t large enough. We have shown the first part, namely, that the algorithm context grows the tree at least as large as to include T_σ.

Consider a past sequence $z = z_1 z_2 \cdots$, relative to $u = x(t)$, such that it lies in the tree $T(t-1)$ between a node in Z_σ and the maximum length node whose counts satisfy the constraint in (11). These are the nodes among which the context $z^*(t)$ is to be selected. We know by the first part of the theorem that for all t greater than some number M such nodes exist. We can expand the second sum in (A1) in Taylor series with the result

$$\Delta(t,z) = P(z) \sum_{y=0,1} P(y/z) \theta^2(t,z,y) + R, \quad \text{(A2)}$$

where $\theta(t, z, y) = P(0/z) - P(0/zy)$ and R consists of the higher order terms such that $R/\theta^2(t, z, y) \to 0$ as $\theta(t, z, y) \to 0$. The estimates $P(0/z) = c_t(0, z)/c_t(z)$ are asymptotically normal with mean $P^0(0/z)$ and standard deviation proportional to $1/\sqrt{c_t(z)}$, where we denote the count of node z in the tree $T(t-1)$ by $c_t(z)$ to emphasize its dependence on t. This is true, because in the source the random variables $x(t), x(t'), \cdots$ that "occur" in a context z, or more correctly, are conditioned on a sequence z which contains a sequence in Z^0 as a subsequence, are independent. Here, the argument then is the same as with Markov chains, where the distinct symbols are dependent but they become independent when they are conditioned on any one state; in other words, "different symbol occurrences at each state are independent," to abuse the language a little. Hence $\theta(t, z, y)$ is also asymptotically normal. Further, by (13) $P^0(u/z) = P^0(u/zy)$, which means that $\Delta^0(z) = 0$, and the mean of $\theta(t, z, y)$ is zero. Its standard deviation $d(t, z, y)$ clearly satisfies the inequality

$$d^2(t,z,y) \leq 1/c_t(z) + \max\{1/c_t(z0), 1/c_t(z1)\} + R' < 2\max\{1/c_t(z0), 1/c_t(z1)\} + 2R', \quad \text{(A3)}$$

where R' denotes the higher order terms such that $R'/d^2(t, z, y) \to 0$ as $d(t, z, y) \to 0$. Further, the ratio $\theta^2(t, z, y)/d^2(t, z, y)$ has asymptotically χ^2 distribution. Using the second bound in (11) with (A3) we have the inequality

$$d^2(t,z,y) < \left(\sqrt{\log t}\right)/\alpha t + 2R' \quad \text{(A4)}$$

for all z of the considered type, all $y = 0, 1$, and all t.

The probability of the event that $z^*(t)$ is not in Z_σ for $t > M$, is by (11) the probability of the event that for at least one z of the above defined type, $\Delta(t, z) > (\log t)/t$. This probability, in turn, is bounded from above by the sum of the probabilities

$$\sum_z \Pr[\Delta(t,z) > (\log t)/t], \quad \text{(A5)}$$

over the at most $\beta \log t$ values of z. From (A2) and (A4) we get the inequalities

$$\begin{aligned} &\Pr[\Delta(t,z) > (\log t)/t] \\ &\quad = \Pr\{\theta^2 + R > (\log t)/t\} \\ &\quad < \Pr\{\theta^2/d^2 > (\alpha - (R+2R')/d^2)\sqrt{\log t}\} \\ &\quad \leq \Pr\{\theta^2/d^2 > \alpha'\sqrt{\log t}\}, \end{aligned}$$

where $\alpha' > 0$ for t greater than some number M'. We also dropped the argument variables for simplicity. The last probability, expressing the probability mass of the tail from $\mu = \alpha'\sqrt{\log t}$ on in the χ^2 distribution, is not greater than $e^{-\mu}$. Therefore, the sum in (A5), which has no more than $\beta \log t$ terms, is not greater than

$$(\beta \log t) e^{-\alpha'\sqrt{\log t}},$$

which goes to zero as t goes to infinity. We have proved (16).

From (8), where $z^*(t)$ denotes the context generated by the rule (11) the ideal per symbol code length is seen to converge to a

conditional entropy (9), where by the first part of the theorem Z contains Z_σ for t greater than some number. The convergence (17) follows with the ergodic theorem. Q.E.D.

References

[1] H. Akaike, "On entropy maximization principle," in *Applications of Statistics*, P. R. Krishnaiah, Ed. Amsterdam: North Holland, 1977, pp. 27–41.

[2] L. D. Davisson, "Comments on 'Sequence time coding for data compression'," in *Proc. IEEE (Lett.)*, vol. 54, p. 2010, Dec. 1966.

[3] G. G. Langdon, Jr., and J. Rissanen, "A simple general binary source code," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 5, pp. 800–803, Sept. 1982.

[4] J. C. Lawrence, "A new universal coding scheme for the binary memoryless source," *IEEE Trans. Inform. Theory*, vol. IT-23, no. 4, pp. 466–472, July 1977.

[5] T. J. Lynch, "Sequence time coding for data compression," in *Proc. IEEE (Lett.)*, vol. 54, pp. 1490–1491, Oct. 1966.

[6] J. S. Ma, "Data compression," Ph.D. dissertation, Dept. Electrical and Computer Engineering, University of Massachusetts, Amherst, 1978.

[7] J. Rissanen, "Tight lower bounds for optimum code length," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 2, pp. 348–349, Mar. 1982.

[8] J. Rissanen and G. G. Langdon, Jr., "Universal modeling and coding," *IEEE Trans. Inform. Theory*, vol. IT-27, no. 1, pp. 12–23, Jan. 1981.

[9] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[10] J. P. M. Schalkwijk, "An algorithm for source coding," *IEEE Trans. Inform. Theory*, vol. IT-18, no. 3, pp. 395–398, 1972.

[11] B. P. Tunstall, "Synthesis of noiseless compression codes," Ph.D. dissertation, Georgia Inst. of Technology, 1968.

[12] J. Ziv and A. Lempel, "Compression if individual sequences via variable-rate-encoding," *IEEE Trans. Inform. Theory*, vol. IT-24, no. 5, pp. 530–536, Sept. 1978.

A Simple Class of Asymptotically Optimal Quantizers

STAMATIS CAMBANIS, MEMBER, IEEE, AND NEIL L. GERR

Abstract—**A simple class of quantizers is introduced which are asymptotically optimal, as the number of quantization levels increases to infinity, with respect to a mean rth power absolute error distortion measure. These asymptotically optimal quantizers are very easy to compute. Their performance is evaluated for several distributions and compares favorably with the performance of the optimal quantizers in all cases for which the latter have been computed. In addition their asymptotic robustness is studied under location, scale, and shape mismatch for several families of distributions.**

I. Introduction

THE quantization of a random variable X with known probability density function $p(x)$ is an important problem which has been studied extensively in the literature. The special issue on quantization published in the March 1982 issue of this TRANSACTIONS provides a comprehensive overview of the problem.

Manuscript received May 11, 1982; revised November 1, 1982. This work was supported by the Air Force Office of Research under Contract AFOSR F49620-82-C-0009. This work was partially presented at the 1982 Conference on Information Sciences and Systems, Princeton, NJ, March 17–19.

S. Cambanis is with the Statistics Department, University of North Carolina, Chapel Hill, NC 27514.

N. L. Gerr was with the Statistics Department of the University of North Carolina at Chapel Hill; he is now with D. H. Wagner, Associates, 3887 Plaza Drive, Fairfax, VA 22030.

Here we consider real valued random variables and mean rth power absolute error distortion measures, so that the optimal N-level quantizer minimizes $\mathcal{E}|X - Q_N(X)|^r$ over all N-level quantizers Q_N. The problem of finding optimal N-level quantizers has been considered by Lloyd [12] and Max [15], and its solution is not generally straightforward. The optimal mean square error N-level quantizers have been computed in the literature for the Gaussian, Rayleigh, and Laplacian distributions using the Lloyd–Max algorithm. (For a discussion of some problems inherent in such algorithms see Bucklew and Gallagher [5].)

In this paper we determine in a very simple manner a sequence Q_N^* of N-level quantizers which, while not necessarily optimal for any N, are nevertheless *asymptotically optimal* in the sense that their rth-order mean approximation error tends to zero as $N \to \infty$ at the same rate as for the sequence of optimal quantizers:

$$\lim_{N\to\infty} \frac{\mathcal{E}|X - Q_N^*(X)|^r}{\inf_{Q_N} \mathcal{E}|X - Q_N(X)|^r} = 1,$$

where the infimum is taken over all N-level quantizers Q_N. Under appropriate conditions on the tails of the density $p(x)$, the asymptotically optimal sequence of quantizers Q_N^* is determined as follows. The quantization levels $a_1^{(N)} < a_2^{(N)} < \cdots < a_N^{(N)}$ are, respectively, the $1/(2N)$, $3/(2N), \cdots, (2N-1)/(2N)$ quantiles of the density $h(x)$

0018-9448/83/0900-0664$01.00