

- [21] D. Slepian, "Prolate spheroidal wave functions, Fourier analysis and uncertainty—IV: Extension to many dimensions; Generalized prolate spheroidal functions," *Bell Syst. Tech. J.*, vol. 43, pp. 3009–3057, Nov. 1964.
- [22] R. J. Glauber, "Photon statistics," in *Laser Handbook*, F. T. Arecchi and E. O. Schulz-DuBois, Eds. New York: Elsevier, 1972, pp. 1–43.
- [23] E. B. Davies, *Quantum Theory of Open Systems*. New York: Academic, 1976, ch. 10.
- [24] H. J. Landau and H. O. Pollak, "Prolate spheroidal wave functions, Fourier analysis and uncertainty—II," *Bell System Tech. J.*, vol. 40, pp. 65–84, Jan. 1961.

Variations on a Theme by Huffman

ROBERT G. GALLAGER, FELLOW, IEEE

Abstract—In honor of the twenty-fifth anniversary of Huffman coding, four new results about Huffman codes are presented. The first result shows that a binary prefix condition code is a Huffman code iff the intermediate and terminal nodes in the code tree can be listed by nonincreasing probability so that each node in the list is adjacent to its sibling. The second result upper bounds the redundancy (expected length minus entropy) of a binary Huffman code by $P_1 + \log_2[2(\log_2 e)/e] = P_1 + 0.086$, where P_1 is the probability of the most likely source letter. The third result shows that one can always leave a codeword of length two unused and still have a redundancy of at most one. The fourth result is a simple algorithm for adapting a Huffman code to slowly varying estimates of the source probabilities. In essence, one maintains a running count of uses of each node in the code tree and lists the nodes in order of these counts. Whenever the occurrence of a message increases a node count above the count of the next node in the list, the nodes, with their attached subtrees, are interchanged.

I. INTRODUCTION

SINCE the appearance 25 years ago of Huffman's [1] classical paper on minimum-redundancy variable-length source coding, Huffman coding has remained one of the most familiar topics in information theory, but has not seen widespread application. One of the difficulties of application arises in the situation where a single source producing nonbinary letters at a fixed rate is to be transmitted over a synchronous binary communication channel. If one uses a variable-length code, then the rate at which binary digits are delivered to the channel will fluctuate, and the expected rate will depend on the relative frequencies of the source letters. This means, first, that buffering is required between the source and channel and, second, that the system will fail if the source statistics become such that the expected rate of binary digits exceeds the channel capacity. For these reasons, and also for

processing simplicity, there has been increasing standardization of fixed-length source codes, for example, the American Standard Code for Information Interchange (ASCII).

The above arguments against variable-length source codes are now much less compelling than before for several reasons. The first is the growth of statistical multiplexors, concentrators, and data networks. Since each of these allocate communication resources to sources on the basis of need, buffering exists as a central part of such systems. Because of the large number of sources, mistaken assumptions about some of the source statistics lead to inefficiency but not failure when using variable-length codes. In addition, such systems require large amounts of protocol, or control information, and the use of fixed-length codes for this control, as, for example, in packet headers, turns out to be very inefficient. Finally, the fact that processing costs and storage costs are dropping very much faster than communication costs has fundamentally changed the trade-off between communication efficiency and processing complexity. For all these reasons one can expect to see much greater use of variable-length codes in the future.

The author has recently been studying possible uses of source coding in data networks, and rather surprisingly, the four quite elementary results described in the abstract turned up. In order to simplify the reader's task as much as possible, we first state and prove each result for the case of binary codewords and then extend the result in the Appendix to arbitrary code alphabets. We start with the sibling property, since that forms the basis for the other results. Section III on redundancy and Section IV on adaptive Huffman coding are independent and can be read in either order.

II. THE SIBLING PROPERTY

Let A be a discrete source with K letters, $2 \leq K < \infty$, and let P_k denote the probability of letter a_k , $1 \leq k \leq K$. It is customary to assume $P_k > 0$, but we allow at most one P_k to be zero in order to simplify examples in which P_k is

Manuscript received December 19, 1977; revised March 23, 1978. This work was supported in part by the National Science Foundation under Grant NSF-ENG76-24447 and in part by Codex Corp., Newton, MA. This paper was presented at the IEEE International Symposium on Information Theory, Cornell University, Ithaca, NY, October 10–14, 1977.

The author is with the Department of Electrical Engineering and Computer Science and the Electronic Systems Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.

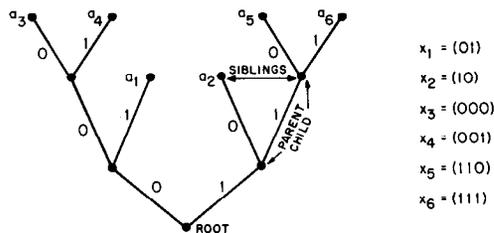


Fig. 1. Rooted binary code tree.

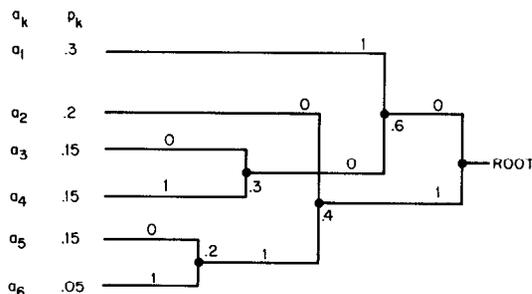


Fig. 2. Huffman encoding algorithm (same code as Fig. 1).

allowed to approach zero arbitrarily closely. Let $x_k = (x_k(1), x_k(2), \dots, x_k(n_k))$ be the binary codeword for letter a_k , $1 \leq k \leq K$. Here $x_k(i)$ is a binary digit, $1 \leq i \leq n_k$, and n_k is called the length of the codeword. A binary code for A is the set of codewords plus the mapping that maps $a_k \rightarrow x_k$, for $1 \leq k \leq K$. A codeword x_k is called a prefix of a codeword x_j if $n_k \leq n_j$ and $x_k(i) = x_j(i)$ for each i , $1 \leq i \leq n_k$. A prefix condition code is a code with the property that no codeword is a prefix of any other codeword. A prefix condition code can be conveniently represented as a rooted binary code tree (see Fig. 1) in which each source letter corresponds to a leaf on the tree and where the associated codeword is the sequence of labels on the path from root to leaf. If two nodes are adjacent on a path from the root to a leaf, we say that the one closer to the root is the *parent* of the other, which is called the *child* of the parent. Two nodes with a common parent are called *siblings*.

It is well-known [2] that an arbitrary concatenation of codewords from a prefix condition code can be uniquely decoded into the corresponding source letters. Furthermore, every code that can be uniquely decoded has lengths satisfying the Kraft inequality

$$\sum_{k=1}^K 2^{-n_k} \leq 1, \tag{1}$$

and prefix condition codes can be constructed with any set of lengths satisfying (1).

Huffman [1] developed an algorithm that generates, for any such source, a prefix condition code that is optimum in the sense of minimizing the expected codeword length

$$E(n) = \sum_{k=1}^K P_k n_k. \tag{2}$$

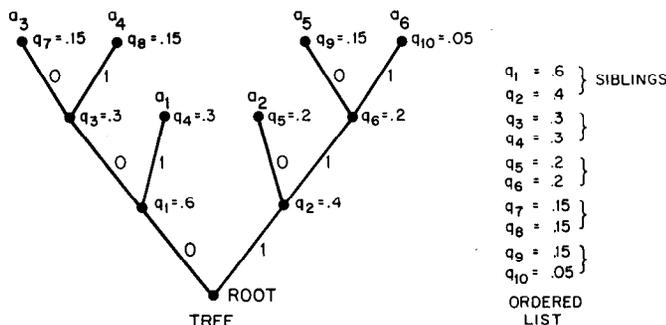


Fig. 3. Sibling property.

The algorithm, illustrated in Fig. 2, is most easily viewed as starting with the leaves of a rooted tree and iteratively generating the intermediate nodes. The algorithm is given below.

- 1) Let L be a list of the probabilities of the source letters corresponding to the leaves of the tree.
- 2) Take two smallest probabilities in L , make the corresponding nodes siblings, generate an intermediate node as their parent, and label the link from parent to one child with zero and the other with one.
- 3) Replace the above two probabilities in L with their sum, associated with the new intermediate node. If the new L contains one element, stop, and otherwise return to step 2).

The codes generated by this algorithm are called Huffman codes. Our first objective is to give a structural characterization of Huffman codes, as opposed to the algorithmic characterization just given.

Each of the leaves of a code tree has a probability assigned to it, namely, the probability of the corresponding source letter. We also assign a probability to each intermediate node, defined recursively as the sum of the probabilities of its children. An equivalent nonrecursive definition is that the probability of an intermediate node is the sum of the probabilities of all leaves for which the path from root to leaf passes through the given intermediate node.

Definition: A binary code tree has the *sibling property* if each node (except the root) has a sibling and if the nodes can be listed in order of nonincreasing probability with each node being adjacent in the list to its sibling.

Fig. 3 illustrates the sibling property. Note that if several nodes have the same probability, as in Fig. 3, the list in order of nonincreasing probability is nonunique, and the definition only requires that there be some such list with each node being adjacent to its sibling. Note also (by a simple inductive argument) that the list (excluding the root) must contain $2K-2$ elements and that for each k , $1 \leq k \leq K-1$, the $2k$ th and $(2k-1)$ th elements on the list must be siblings for the sibling property to hold.

Theorem 1: A binary prefix condition code is a Huffman code iff the code tree has the sibling property.

Proof: First assume that a code tree has the sibling property. Then the last two elements on the ordered list are siblings and, in addition, must be leaves, for if one were an intermediate node, at least one of the children of that intermediate node would have a smaller probability than the intermediate node,¹ which is impossible because of the ordering property. Thus these nodes correspond to two smallest probability source letters and can be made siblings in the first execution of step 2) in the Huffman algorithm. Now remove these siblings from the code tree, removing also the last two elements from the ordered list. The resulting reduced code tree still has the sibling property, and the leaves of the reduced code tree correspond to the list L in the Huffman algorithm after the first execution of step 3). Thus the above argument can be repeated: at each step the Huffman algorithm chooses, as siblings, elements which are siblings in the original code tree. By matching the link labels in the Huffman code to those in the original code tree, the two codes are seen to be identical. Next assume that a binary code tree is generated by the Huffman algorithm, and assume that each time the algorithm executes step 2), we add the two nodes defined as siblings to the top of an initially empty list, putting the less probable below the more probable. The list so generated clearly has each node adjacent to its sibling, so to establish the sibling property, we simply have to show that the list is nonincreasing in order of probability. This is trivial, however, since at each iteration the two elements added to the list have probabilities less than or equal to that of each element in the new L of the Huffman algorithm, and the next two elements added to the list are chosen from this new L . Q.E.D.

Next define the *level* of a node as the number of links on the path from the root to the node. It is clear from the optimality of Huffman codes that for each $l \geq 1$ the probability of each node at level l is less than or equal to the probability of each node at level $l-1$. For the purist, this property can be directly derived from the sibling property using induction on l . Define an *ordered* Huffman code as a Huffman code in which when two nodes are defined as siblings, the label zero is assigned to the link going to the more probable of the siblings. Also define a *lexicographically ordered* code tree as a tree in which, for each $l \geq 1$, the probability of each node at level l is less than or equal to the probability of each node at level $l-1$ and in which the probabilities of nodes at level l are monotonically nonincreasing in the binary number corresponding to their path names from the root.

Corollary: A binary prefix condition code is an ordered Huffman code iff the code tree is lexicographically ordered.

Proof: Lexicographic ordering implies the sibling property, which implies that the code is a Huffman code. Lexicographic ordering also implies that the link from a

parent to the more probable of the siblings is labelled zero, implying an ordered Huffman code. Now assume an ordered Huffman code, and use induction on the level l . The nodes at level 1 are lexicographically ordered by construction. Assume for any $l > 1$ that the nodes at level $l-1$ are lexicographically ordered. By the sibling property, if the probability of one parent is greater than or equal to that of another, the children are correspondingly ordered. This shows that nonsibling nodes at level l have the correct ordering. Siblings, however, are correctly ordered by the construction. Q.E.D.

III. THE REDUNDANCY OF HUFFMAN CODES

The redundancy r of a source code is defined to be the expected length of the codewords minus the binary entropy $H(P_1, \dots, P_K)$ of the source probabilities

$$r = \sum_{k=1}^K P_k n_k - H(P_1, \dots, P_K) \quad (3)$$

where $H(P_1, \dots, P_K) = -\sum_k P_k \log P_k$. It is well-known [2] that for optimal codes the redundancy always lies between zero and one. The upper limit, one, is reached by a source with two letters of probabilities zero and one, or more strictly, is approached as $\epsilon \rightarrow 0$ by a source with probabilities $1-\epsilon$ and ϵ . Our purpose here is to show that when the most probable letter in a source has a probability much less than one, then the upper limit on r can be greatly improved upon.

Suppose we have a Huffman code, and using the sibling property, we number all of the nodes (except the root node) in order of decreasing probability and increasing level so that for each k , $1 \leq k \leq K-1$, nodes $2k$ and $2k-1$ are siblings. Let q_k be the probability of the k th node on the list, $1 \leq k \leq 2K-2$. The expected length of the code can be written as

$$E(n) = \sum_{k=1}^{2K-2} q_k \quad (4)$$

In order to see this, perform the conceptual experiment of writing each q_k in (4) as the sum of the leaf probabilities of leaves whose path from the root passes through k . Then a codeword i of length n_i has its probability P_i written in n_i of these sums, showing the equivalence of (4) and (2). We can also rewrite the entropy as

$$H(P_1, \dots, P_K) = \sum_{k=1}^{K-1} (q_{2k-1} + q_{2k}) \mathcal{H} \left(\frac{q_{2k}}{q_{2k-1} + q_{2k}} \right) \quad (5)$$

where \mathcal{H} is the binary entropy function

$$\mathcal{H}(x) = -x \log_2 x - (1-x) \log_2 (1-x). \quad (6)$$

Each term in (5) is the probability that a given parent in the code tree will occur times the entropy of the choice of its child. Formally (5) can be established by induction on reduced trees. Combining (5) and (4), we have

$$r = \sum_{k=1}^{K-1} (q_{2k-1} + q_{2k}) \left[1 - \mathcal{H} \left(\frac{q_{2k}}{q_{2k-1} + q_{2k}} \right) \right]. \quad (7)$$

¹This is where we use the assumption that at most one source letter have zero probability. The theorem is true without this restriction, but the proof is harder and the restriction is of no importance.

Finally, let $l \geq 1$ be some level at which the tree is full (i.e., for which the tree has $L=2^l$ nodes of level l), but for which there are also nodes at level $l+1$. For $K > 2$ such an l must exist. Let m be the smallest integer for which node $2m-1$ is at level $l+1$, and let q'_1, \dots, q'_L be the probabilities of the nodes at level l . Splitting the sum in (7) into $k < m$ and $k \geq m$ and writing the terms for $k < m$ in the form of (3), we have

$$r = l - H(q'_1, \dots, q'_L) + \sum_{k=m}^{K-1} (q_{2k-1} + q_{2k}) \cdot \left[1 - \mathcal{H}\left(\frac{q_{2k}}{q_{2k-1} + q_{2k}}\right) \right]. \quad (8)$$

For $K=2$ (8) is valid for $l=1$ with the final term omitted.

Theorem 2: Let P_1 be the probability of the most likely letter in a finite discrete source. Then the redundancy of the Huffman code for the source satisfies

$$r \leq P_1 + \sigma \quad (9)$$

where $\sigma = 1 - \log_2 e + \log_2(\log_2 e) \approx 0.086$. For $P_1 \geq 1/2$,

$$r \leq 2 - \mathcal{H}(P_1) - P_1 \leq P_1. \quad (10)$$

Proof: For $0 \leq x \leq 1/2$, $\mathcal{H}(x) \geq 2x$. Using this inequality, the final term in (8) is upper bounded by

$$\sum_{k=m}^{K-1} q_{2k-1} - q_{2k}.$$

Since the sequence q_k is nonincreasing, this is further upper bounded by q_{2m-1} so that

$$r \leq l - H(q'_1, \dots, q'_L) + q_{2m-1}. \quad (11)$$

Let n_1 be the length of the shortest codeword, which must correspond to a source letter of probability P_1 . First assume $P_1 \geq 1/2$. Then $n_1 = 1$, and taking $l=1$ in (11), we obtain

$$r \leq 1 - \mathcal{H}(P_1) + q_{2m-1}. \quad (12)$$

Since $q_{2m-1} \leq 1 - P_1$, (12) implies the first inequality in (10). The second inequality is satisfied with equality at $P_1 = 1/2$ and $P_1 = 1$ and is satisfied in between because of convexity. This also establishes (9) for $P_1 \geq 1/2$ and thus whenever $K=2$. If all the codewords are of the same length $n_1 > 1$, choose l in (8) to be $n_1 - 1$, and otherwise choose $l = n_1$. In both cases $q_{2m-1} \leq P_1$, and in both cases we can order q'_1, \dots, q'_L to satisfy $q'_1 \geq q'_2 \geq \dots \geq q'_L \geq q'_1/2$. Let Q be the set of choices for q'_1, \dots, q'_L that satisfy the above linear inequality constraints along with $\sum q'_i = 1$. Then

$$r \leq l - \min_{\{q'_i\} \in Q} H(q'_1, \dots, q'_L) + P_1. \quad (13)$$

Since H is convex, the minimum above must occur at an extreme point of Q ; the extreme points of Q are those for which for some n , $1 \leq n \leq L$, $q'_i = q'_1$, for $i \leq n$, and $q'_i = q'_1/2$, for $n < i \leq L$. For a given n , then, $q'_1 = 2/(L+n)$, and

$$\min_{\{q'_i\} \in Q} H(q'_1, \dots, q'_L) = \min_{1 \leq n \leq L} \left[-\log \frac{2}{L+n} + \frac{(L-n)}{L+n} \right].$$

Further lower bounding by allowing n to take on noninteger values, we have an elementary calculus problem with the minimum value $l - \sigma$, where $\sigma = 1 - \log_2 e + \log_2(\log_2 e)$. Substituting this in (13) completes the proof.

The bound on redundancy here is quite tight. For all $P_1 \geq 1/2$, a source with probabilities $(P_1, 1 - P_1, 0)$ satisfies the bound with equality. Also, the source with probabilities $(1/3, 1/3, 1/3, 0)$ has a redundancy of 0.415, whereas the bound for $P_1 = 1/3$ is 0.419. It can also be shown by tedious calculation that for all P_1 there exist sources with $r \geq \sigma$, which shows that the bound is tight in the limit $P_1 \rightarrow 0$.

Next we take a somewhat different approach to redundancy. Suppose we want to reserve one or more codewords for control or protocol purposes. One could regard these control messages as having probabilities like everything else, but it is frequently more convenient to regard them separately.

Theorem 3: For every finite discrete source, there exists a prefix condition code with an unused codeword of length two and with redundancy $r \leq 1$.

Proof: We shall construct the desired code by first constructing the Huffman code for the source. We then take the less probable node at level one, let us say node two, and move it to level two, making it the sibling of a newly created reserved word and making the parent the sibling of the other level-one node; all other parent-children relationships are unchanged. In effect we have lengthened by one each codeword stemming from node two. Let q_1 and q_2 be the probabilities of the original level-one nodes $q_1 \geq q_2$. Then if r is the redundancy of the original code and r' is that of the modified code, $r' = r + q_2$. From (11), using $l=1$, we have

$$\begin{aligned} r &\leq 1 - \mathcal{H}(q_2) + q_2 \\ r' &\leq 1 - \mathcal{H}(q_2) + 2q_2 \leq 1 \end{aligned}$$

where in the final inequality we have used the fact that $q_2 \leq 1/2$.

Note that for a source with probabilities $(1/2, 1/2, 0)$ this bound is met with equality. It also can be shown that the above procedure for choosing an unused word of length two is optimal in the sense of minimizing r' .

IV. ADAPTIVE HUFFMAN CODES

In this section we are interested in Huffman encoders which maintain a running estimate of the source letter probabilities; as these estimates change, the code will change, remaining optimal for the current estimates. Our primary concern is with the algorithm to modify the code rather than the problem of estimating the probabilities. In fact, the method to be used to estimate the letter probabilities is almost trivial. Simply maintain a counter for each letter of the source alphabet, and increment the counter each time that letter occurs. Periodically, let us say after each N th letter in the source sequence, multiply each of

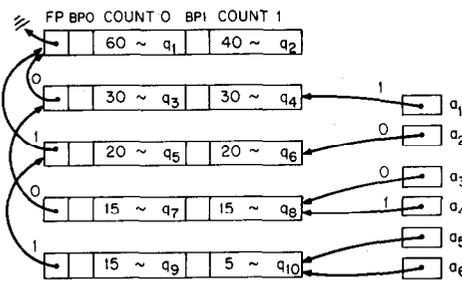


Fig. 4. Data structure for adaptive Huffman coder.

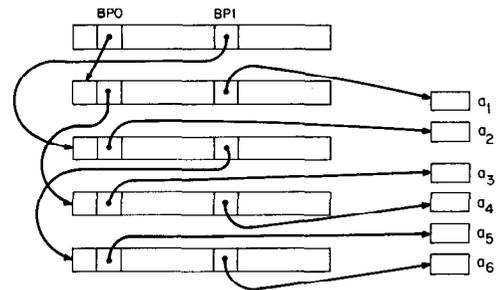


Fig. 5. Backward pointers for structure of Fig. 4.

the letter counts by some fixed number $\alpha < 1$. The current estimate for the probability of a source letter is the current count for that letter divided by the sum of all the counts. Since the counts are proportional to the probability estimates, the algorithm to be described operates directly on the counts and never needs to calculate the probability estimates.

The choice of α and N determine how quickly the estimates can change, and it can be seen that the time constant for the number of letters entering into the estimate is $N/(1-\alpha)$. As this time constant increases, the adaptation becomes slower, yielding better estimates for slowly varying statistics, but more irrelevant estimates for rapidly varying statistics. For small time constants, of course, the estimates will be noisy, whether the statistics are slowly or quickly varying. For a given time constant, it seems appropriate to choose $\alpha = 1/2$, since this makes the multiplication by α simple and keeps N relatively large.

The algorithm will keep a count for each node in the current code tree. By the sibling property, if the nodes can be listed by decreasing counts so that each node is adjacent to its sibling, then the code is optimal for the current probability estimates.

We can implement this strategy, in a microcomputer let us say, by maintaining a fixed list of sibling pairs in storage. For a K -letter alphabet, there are $K-1$ such sibling pairs. The storage location for each sibling pair will contain five components, two of which are the current counts for the sibling nodes and three of which are pointers to be described later. The structure will be maintained in such a way that each count for the top sibling pair will be greater than or equal to each count for the next pair, and so forth, down to the bottom of the list.

The structure of the code tree is maintained in the sibling list by a set of forward pointers, indicated by FP in Fig. 4. The FP pointer for a given sibling pair points to the parent of the pair, or more strictly, to the sibling pair containing the parent, with an extra bit to indicate whether the parent is the zero sibling or the one sibling. The source letters themselves have a separate storage area containing only a pointer to the letter's current location in the sibling list. For example, in Fig. 4 if a letter a_1 occurred, the last digit of codeword x_1 would be determined as one, since the pointer from a_1 goes to the one side of the second sibling pair. Since the FP pointer from this pair goes to the zero side of the first sibling pair, the

codeword is determined as 01. The first sibling pair always corresponds to the level-one nodes, and its FP pointer is nil.

We have now seen how codewords are generated and next take up the problem of updating the counts and perhaps changing the code. It is important to recognize that a codeword must be generated *before* any changes to the code are allowed, since the decoder must decode the codeword to obtain the information on which changes are based. After generating a codeword, the counts are incremented one by one on the nodes on the path from leaf node to root. For the example of letter a_1 , first the one count in the second sibling pair is incremented from 30 to 31 and then the zero count in the first sibling pair is incremented from 60 to 61. Each time a count is incremented, the count must be compared with the counts of the next higher sibling pair, and if it exceeds one of these counts, the two nodes must be interchanged, which means that the forward pointers into those nodes must be switched. The purpose of the backward pointers, BPO and BPI in Fig. 5, is to find these forward pointers without a search. These pointers also allow the decoder to decode easily. We see that a code change is made by changing two FP pointers and two BP pointers. The point of changing the counts one at a time is that if a code change is made when one count is changed, it is the new path to the root rather than the old path that must have its counts incremented. For example, if a_5 occurs, then a_3 and a_5 would be interchanged in the tree. The zero count in the *fourth* sibling pair would become 16, the zero count in the *second* sibling pair would become 31, and the zero count of the first would become 61.

It is possible for several code changes to occur for one source digit, but there is at most one change for each count incremented and thus at most one change for each encoded bit. The entire computation corresponding to an encoded bit is then one memory access to find the bit and the location of the next bit, one memory access to find the count to be incremented, two accesses and comparisons to see if the code must be changed, and four pointer changes if the code is changed (it is not necessary to interchange the counts, since a comparison for equality can be done before the incrementing, and then the new count can be incremented). Thus, in summary, the computational load is independent of the alphabet size and proportional to the code bit generation rate.

In the above description, we have left out one annoying detail. Many counts in the list could be the same. If one of those counts were to be incremented, the interchange would have to be with the first sibling pair containing that number. The search to find this first sibling pair can be avoided, at the expense of storage, by having a storage location for each such list. Each element on the list has a pointer to that location, and that location contains a pointer to the top of the list (which could contain just one element). Since elements join such a list from the bottom (in terms of the ordered sibling pairs) and leave from the top (after the code switch), no variable-length searches are required.

APPENDIX

We now generalize the previous results to the case where the code alphabet is of arbitrary size D rather than binary. We allow $D-1$ of the source messages to have zero probability; because of this, we can assume without loss of generality that the size of the message set K is $c(D-1)+1$ for some integer c . The Huffman coding algorithm is then changed by replacing the word "two," in the algorithm as given, by the letter D . A code tree has the *sibling property* if the nodes, excluding the root, can be listed in order of nonincreasing probability such that for each i , $1 \leq i \leq c$, nodes $iD, iD-1, \dots, iD-D+1$ are all siblings of each other. The proof of Theorem 1, that a prefix condition code is a Huffman code iff the code tree has the sibling property, is the same as the original proof, with "two" replaced by D . The lexicographic property also follows in an obvious fashion.

We define the redundancy r of a source code with a code alphabet size of D as the expected codeword length minus the entropy, in base D , of the source probabilities. If we let l be some level at which the code tree is full, let $L = D^l$ nodes, number the nodes in order of nonincreasing probability, and let m be the smallest integer for which node $Dm - (D-1)$ is at level $l+1$, then, as in (8), we have

$$r = l - H(q'_1, \dots, q'_L) + \sum_{k=m}^c t_k [1 - H(\vec{q}_k)] \quad (\text{A1})$$

when q'_i , $1 \leq i \leq L$, is the probability of the i th node in level l , $t_k = q_{Dk} + q_{Dk-1} + \dots + q_{Dk-(D-1)}$, and \vec{q}_k is a probability vector with components $(q_{Dk}/t_k, q_{Dk-1}/t_k, \dots, q_{Dk-D+1}/t_k)$. The final term is zero if all terminal nodes have level l .

The final term in (A1) is more difficult to handle than the corresponding term in the binary case. We need the following lemma.

Lemma: Let $x_1 \geq x_2 \geq \dots \geq x_D$ be probabilities, $\sum_{i=1}^D x_i = 1$, and let $H(x_1, \dots, x_D)$ be the entropy base D . Then

$$1 - H(x_1, \dots, x_D) \leq (x_1 - x_D)D / \ln D \quad (\text{A2})$$

Proof: By definition,

$$1 - H(x_1, \dots, x_D) = \sum_{i=1}^D x_i \log_D(x_i D).$$

Choose λ_i , $1 \leq i \leq D$, such that $x_i = \lambda_i x_1 + (1 - \lambda_i) x_D$. Then

$$\begin{aligned} \sum_{i=1}^D x_i \log_D(x_i D) &\leq \sum_{i=1}^D \lambda_i x_1 \log_D(x_1 D) \\ &\quad + \sum_{i=1}^D (1 - \lambda_i) x_D \log_D(x_D D). \end{aligned} \quad (\text{A3})$$

Using the fact that $\sum x_i = 1$, we see that $\sum \lambda_i = (1 - x_D D) / (x_1 - x_D)$, and the right side of (A3) is equal to

$$\frac{(1 - x_D D) x_1}{x_1 - x_D} \log_D(x_1 D) + \frac{(x_1 D - 1) x_D}{x_1 - x_D} \log_D(x_D D).$$

Since $x_D D \leq 1$ and $x_1 D \geq 1$, we can upper bound this with the inequality $\log_D x \leq (x-1)/\ln D$:

$$\begin{aligned} \sum_{i=1}^D x_i \log_D(x_i D) &\leq (1 - x_D D)(x_1 D - 1) / \ln D \\ &\leq (1 - x_D D) x_1 D / \ln D \leq (x_1 D - x_D D) / \ln D. \end{aligned}$$

It is likely that the bound in (A2) could be improved somewhat, but it is necessary that the bound increase with increasing D as $D/\ln D$. Substituting (A2) into (A1), we obtain

$$r \leq l - H(q'_1, \dots, q'_L) + q_{mD-(D-1)} D / \ln D. \quad (\text{A4})$$

Assuming that level $l+1$ is nonempty, $q'_L \geq q'_1/D$ for an optimum code, and we can lower bound the entropy term in (A4), as in Theorem 2, by

$$H(q'_1, \dots, q'_L) \geq l - \sigma_D \quad (\text{A5})$$

$$\sigma_D = \log_D(D-1) + \log_D(\log_D e) - \log_D e + \frac{1}{D-1}. \quad (\text{A6})$$

Thus the generalization of Theorem 2 is given by

$$r \leq \sigma_D + P_1 D / \ln D. \quad (\text{A7})$$

Unfortunately, as D gets large, $\sigma_D \rightarrow 1$, but the approach is not rapid. For example, $\sigma_3 = 0.135$, $\sigma_5 = 0.194$, $\sigma_{10} = 0.269$, $\sigma_{20} = 0.335$. As in the case of Theorem 2, for any given D there are sources with P_1 arbitrarily small, for which r is arbitrarily close to σ_D .

Next we extend Theorem 3 by showing that with a code alphabet size D it is always possible to have $D-1$ unused codewords of length two while still maintaining a redundancy $r' \leq 1$. The strategy is the same as before: construct a Huffman code for the source, and then lengthen by one each codeword emanating from the least likely level-one node, leaving $D-1$ unused level-two nodes. Let $\vec{q} = (q_1, q_2, \dots, q_D)$ be the probabilities of the level-one nodes $q_1 \geq q_2 \geq \dots \geq q_D$, and let r be the redundancy of the original code. Then $r' \leq r + q_D$. By considering a line from the probability vector $(1/D, 1/D, \dots, 1/D)$ through \vec{q} to the point \vec{q}' where $q'_D = 0$ and using convexity, we find that

$$H(\vec{q}) \geq D q_D. \quad (\text{A8})$$

Substituting this into (A4) at $l=1$, we have

$$r \leq 1 - D q_D + D q_D / \ln D \quad (\text{A9})$$

$$r' \leq 1 - D q_D (1 - 1/(\ln D) - 1/D). \quad (\text{A10})$$

The term in parentheses is positive for $D \geq 4$, leaving us only the case $D=3$ to consider. First assume that the Huffman code tree has only level-one nodes. Then the final term in (A9) vanishes, and $r' \leq 1$. Next we use (A1) for $l=0$, using the lemma to bound all nodes on the third level or more:

$$r \leq [1 - H(\vec{q})] + \sum_{k=1}^3 q_k [1 - H(\vec{q}_k)] + 3q'/\ln 3 \quad (\text{A11})$$

where q' is the probability of the most probable third-level node and is zero if no third-level nodes exist and where \vec{q}_k is the set of conditional probabilities for the second-level nodes emanating from the k th first-level node; if no second-level nodes emanate from node k , we take $[1 - H(\vec{q}_k)] = 0$.

First assume that third-level nodes exist. Then $q' \leq 1/9$, and for each first- and second-order node, the least likely sibling has at least $1/D$ the probability of the most likely so that (A5)

applies. Thus

$$r \leq 2\sigma_3 + (3/9)/\ln 3.$$

Since $q_3 \leq 1/3$, this implies $r' \leq 1$. Finally, assume $q' = 0$, and let k' be the highest number first-level node for which second-level nodes exist. For $k < k'$, (A5) applies if second-level nodes exist for node k , and

$$r' \leq \sigma_3 + \sum_{k=1}^{k'-1} q_k \sigma_3 + q_{k'} + q_3. \quad (\text{A12})$$

Since second-level nodes exist by assumption, $q_1 \leq q_3/3$ so that $q_1 + q_3 \leq 0.8$. Using this it is easy to verify that $r' \leq 1$ for $k' = 1, 2, 3$, completing the proof.

Adaptive Huffman coding for an alphabet size of $D > 2$ is essentially the same as for $D = 2$. For a source alphabet of

$K = c(D-1) + 1$ letters, we need c storage locations, each containing $2D + 1$ components; there are D counts, one for each sibling, D backward pointers, and one forward pointer. For $D > 2$, and perhaps also for $D = 2$, it is desirable to keep the counts ordered within the storage locations, thus requiring only one comparison instead of D to see if the code must be changed for each encoded letter. The trade-off here is between number of comparisons and number of code changes, which is really an implementation detail.

REFERENCES

- [1] D. A. Huffman, "A method for the construction of minimum redundancy codes," *Proc. IRE*, vol. 40, pp. 1098-1101, 1952.
- [2] R. G. Gallager, *Information Theory and Reliable Communication*, New York: Wiley, 1968.

A Unified Approach to Weak Universal Source Coding

JOHN C. KIEFFER

Abstract—A new method of constructing a universal sequence of block codes for coding a class of ergodic sources is given. With this method, a weakly universal sequence of codes is constructed for variable-rate noiseless coding and for fixed- and variable-rate coding with respect to a fidelity criterion. In this way a unified approach to weak universal block source coding is obtained. For the noiseless variable-rate coding and the fixed-rate coding with respect to fidelity criterion, the assumptions made on the alphabets, distortion measures, and class of sources are both necessary and sufficient. For fixed-rate coding with respect to a fidelity criterion, the sample distortion of the universal code sequence converges in L^1 norm for each source to the optimum distortion for that source. For both variable-rate noiseless coding and variable-rate coding with respect to a fidelity criterion, the sample rate of the universal code sequence converges in L^1 norm for each source to the optimum rate for that source. Using this fact, a universal sequence of codes for fixed-rate noiseless coding is obtained. Some applications to stationary nonergodic sources are also considered. The results of Davisson, Ziv, Neuhoff, Gray, Pursley, and Mackenthun are extended.

I. INTRODUCTION

LET (A, \mathcal{Q}) be a measurable space. (A will serve as the source alphabet.) Let $(A^\infty, \mathcal{Q}^\infty)$ be the measurable space consisting of A^∞ , the set of all doubly-infinite sequences $(\dots, x_{-1}, x_0, x_1, \dots)$ from A , and \mathcal{Q}^∞ , the usual

product σ -field of subsets of A^∞ . For each integer i , let X_i be the i th projection from $A^\infty \rightarrow A$; that is, if $x = (\dots, x_{-1}, x_0, x_1, \dots) \in A^\infty$, $X_i(x) = x_i$. For each $n = 1, 2, \dots$, (A^n, \mathcal{Q}^n) is the measurable space consisting of A^n , the set of all n -tuples (x_1, x_2, \dots, x_n) from A , and \mathcal{Q}^n , the usual product σ -field. For each $n = 1, 2, \dots$, $X^n: A^\infty \rightarrow A^n$ is the projection such that if $x = (\dots, x_{-1}, x_0, x_1, \dots) \in A^\infty$, then $X^n(x) = (x_1, x_2, \dots, x_n)$. A source μ is a probability measure on \mathcal{Q}^∞ . A stationary source is a source stationary with respect to the two-sided shift transformation on A^∞ . An ergodic source is a source stationary and ergodic with respect to the shift.

By means of a block code one can code block by block each sequence in A^∞ generated by a source. Various types of block codes have been considered in the literature: variable-rate codes for noiseless coding [1], fixed-rate codes for coding with respect to a fidelity criterion [2], and variable-rate codes for coding with respect to a fidelity criterion [3]. Let \mathcal{C} be a class of codes of any one of these types. A code $\phi \in \mathcal{C}$ is of order n if ϕ codes a sequence x in A^∞ by subdividing x into blocks of length n and then coding each block individually. Let Λ be a class of ergodic sources. The purpose of universal coding is to find a universal sequence of codes for Λ ; that is, a sequence $\{\phi_n\}_{n=1}^\infty$ from \mathcal{C} such that

- a) ϕ_n is of order n , $n = 1, 2, \dots$,
- b) if $\mu \in \Lambda$, then as $n \rightarrow \infty$, ϕ_n codes μ optimally.

Manuscript received September 23, 1977; revised April 26, 1978.

The author is with the Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305, on leave from the Department of Mathematics, University of Missouri, Rolla, MO 65401.