have, for the second term of the right-hand side of (A10)

$$\sum_{j=2}^{\infty} \frac{G_j - 1}{G_j} (\log G_j) \frac{1}{F_{j-1}} \geq \sum_{j=2}^{\infty} 1 = \infty$$

from which together with the fact that the third term of the right-hand side of (A10) is the sum of the positive numbers, it follows that $E = \infty$. Hence, in view of (A3)–(A5), we have $H(P) = \infty$.

ACKNOWLEDGMENT

The authors wish to thank H. Yamamoto of the University of Tokyo, who brought related references to their attention.

REFERENCES

[1] J. Abrahams, "Huffman-type codes for infinite source distributions," *J. Franklin Inst.*, vol. 331B, no. 3, pp. 265–271, 1994.
[2] R. G. Gallager and D. C. Van Voorhis, "Optimal source coding for geometrically distributed integer alphabets," *IEEE Trans. Inform. Theory*, vol. IT-21, no. 2, pp. 228–230, 1975.
[3] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 10, pp. 1098–1101, 1952.
[4] P. A. Humblet, "Optimal source coding for a class of integer alphabets," *IEEE Trans. Inform. Theory*, vol. IT-24, no. 1, pp. 110–112, 1978.
[5] B. Montgomery and J. Abrahams, "On the redundancy of optimal binary prefix-condition codes for finite and infinite sources," *IEEE Trans. Inform. Theory*, vol. IT-33, no. 1, pp. 156–160, 1987.

Extended Synchronizing Codewords for Binary Prefix Codes

Wai-Man Lam and Sanjeev R. Kulkarni, *Member, IEEE*

***Abstract*—Synchronizing codewords (SC's) have been previously studied as a means to stop error propagation in variable-length codes. However, SC's retain one disadvantage: the symbols after the SC may be put in the wrong positions since the number of decoded symbols before the SC can be different from the original number due to channel errors. Thus we propose the idea of extended synchronizing codewords (ESC's) which can overcome the drawback of SC's. After the decoder receives an ESC, the decoder correctly knows it is in synchronization, regardless of the preceding slippage. We derive some of the essential properties of ESC's and provide several upper bounds on the amount of overhead needed in designing a code with an ESC.**

***Index Terms*—Variable-length coding, prefix codes, errors, synchronization, synchronizing codewords, markers.**

I. INTRODUCTION

One of the most commonly used entropy coding techniques is Huffman coding. Huffman codes are instantaneous prefix codes since no codeword is a prefix of other codewords, and each codeword can be decoded as soon as it is complete. When Huffman codes are transmitted through a noisy channel, a single bit error can lead to loss of synchronization at the decoder and can cause error propagation. Earlier work (e.g., see [2]–[4]) has shown that error propagation can be stopped by using synchronizing codewords (SC's). A synchronizing codeword has the property that after the decoder has decoded it, re-synchronization will take effect regardless of any preceding synchronization slippage.

The problems of synchronization in variable-length coding and the use of synchronizing codewords in particular have been well-studied. For example, Ferguson and Rabinowitz [3] give sufficient conditions for a source to admit a binary synchronous Huffman code. Montgomery and Abrahams [4] have developed a procedure for constructing suboptimal synchronous binary codes for gapless sources. Capocelli *et al.* [2] propose an algorithm to construct synchronizing codewords over arbitrary alphabets and arbitrary probability distributions.

However, SC's retain one significant disadvantage. Although the decoder will be synchronized after decoding an SC, the decoded symbols after the SC may be shifted since the number of decoded symbols before the SC may be different from the original number (due to decoding a variable-length code in the presence of errors). Also, even though the decoder will be synchronized after decoding a synchronizing codeword, the decoder may not realize it had previously been out of synchronization or that is has re-established synchronization.

In this correspondence, we study the design of variable-length codes that allow a more robust decoding in the presence of channel errors. We introduce the concept of *extended synchronizing codewords* (ESC's) which can overcome the drawbacks of SC's. Specifically, an ESC allows both detection and correction of loss of synchronization. This also allows an ESC to be used as a marker in the bit stream to prevent propagation of both decoding errors and symbol shift errors. Thus ESC's can guarantee both codeword and symbol synchronization, so that the decoded symbols after the ESC will not only be decoded correctly, but will also be put in their correct positions. Applications of this idea to image coding have been studied in [5]–[7]. The use of a special codeword as a marker for synchronization recovery is suggested in [5] and [6], where the terms "unique codeword" and "clear codeword" are used, respectively. Such codewords consist of a unique pattern of bits that cannot be formed by any concatenation of codewords. The notion of an ESC in this correspondence formalizes the definition of unique codewords, and we will also show some of the basic capabilities and limitations in using such codewords as markers in a bit stream. In particular, we will show that one of the properties of an ESC, that follows naturally its definition, is that it cannot be formed from any concatenation of codewords. We also obtain results on the amount of overhead needed in designing a code with an ESC.

One of the important differences between an SC and an ESC is that an SC is used as a signal symbol for the variable-length codebook. The codebook can also be designed in a way that no overhead is necessary and the codebook still contains at least one SC. On the other hand, an ESC can be used in two different ways. First, it can be used as a signal element just like an SC. This allows an additional benefit over an SC in that the errors can be detected and decoding re-synchronization can take place. A decoder can then, if desired, invoke algorithms for concealment upon the detection of errors. A second use of an ESC is as an additional symbol placed

Manuscript received October 29, 1994; revised July 27, 1995. This work was supported in part by the National Science Foundation under NYI Grant IRI-9457645.

W.-M. Lam is with Thomson Multimedia, Indianapolis, IN 46290 USA.
S. R. Kulkarni is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA.

Publisher Item Identifier S 0018-9448(96)02926-4.

in regular positions as a marker in the bit stream. Typical examples of this include the use of an ESC for an End-Of-Line (EOL) or an End-Of-Block (EOB) symbol in image coding. This allows symbol synchronization as well as decoding synchronization. However, note that the symbol synchronization capability is brought about by the error-resilient structure of the transmission method (i.e., the use of a regular marker) together with the properties of the ESC. On the other hand, with self-synchronizing codes or in codes using an ESC as a signal element no structure is imposed on the transmission method, and only decoding synchronization without symbol synchronization is possible. Overhead is necessary in constructing a codebook with an ESC whether or not the ESC is used as a signal symbol. Thus a codebook with an ESC will lose some transmission efficiency compared with one which does not have any ESC.

In Section II, we provide some preliminary definitions and notation. In Sections III and IV, we derive some of the essential properties of ESC's. In particular, in Section IV, we prove some upper bounds on the amount of overhead in designing a binary prefix code with at least one ESC. Section V concludes the paper.

II. Preliminary Definitions and Notation

Let X be the set of binary symbols $\{0, 1\}$, and X^n be the set of all sequences obtained by concatenating n symbols of X. Let $X^+ = \bigcup_{n\geq 1} X^n$ be the set of all finite sequences of elements of X. Let $X^* = X^+ \cup \{\lambda\}$ where λ is the empty sequence. A finite subset C of X^+ is called a *code*, and every $c \in C$ is called a codeword.

If a codeword $c = ps$ for some p, $s \in X^*$, then p is called a prefix of c, and s is called a suffix of c. For a given codeword $c \in C$, the sets of all prefixes and suffixes of c are denoted by prefix(c) and suffix(c), respectively, and prefix(C) and suffix(C) denote the set of all prefixes and suffixes, respectively, of all codewords in C. A binary string c_i is a substring of c_j if $c_j = pc_is$ where $p \in$ prefix(c_j) and $s \in$ suffix(c_j). A code is called a prefix code if no codeword is a prefix of any other codeword, in which case we say the codewords satisfy the prefix condition. We consider only binary prefix codes in this correspondence.

For a codeword $c \in C$, let $l(c)$ denote the length of c, and let $L = \max\{l(c)|c \in C\}$. A length vector $\alpha = (\alpha_1, \cdots, \alpha_L)$ of a code C is a vector where α_i is the number of codewords of length i in C. For an i.i.d. source S with symbols $(s_1, \cdots, s_n)$, we denote the probability distribution of the symbols by $(p_1, \cdots, p_n)$. A code $C = \{c_1, \cdots, c_n\}$ is associated with the source S if c_i is assigned to the symbol s_i. The average codeword length of C is given by

$$E(C) = \sum_{i=1}^{n} p_i l(c_i).$$

Huffman codes for S are defined as codes which are obtained using the Huffman algorithm. Huffman codes for S are optimal in the sense that they minimize the average codeword length among all prefix codes for the source S.

III. Definition and Basic Properties of Extended Synchronizing Codewords

Before we provide a definition of an extended synchronizing codeword, consider the possible ways that a particular codeword c_0 can be decoded incorrectly if there are errors preceding c_0, and c_0 has no error. There are two possibilities of error as shown in Fig. 1., where the shaded rectangle represents the codeword c_0. The first possibility occurs when a prefix of c_0 is decoded as a suffix of some other codeword, and a suffix of c_0 is decoded as a prefix of some other codeword (with perhaps some other complete codewords decoded in between). The second possibility occurs when the decoder concatenates c_0 with a binary string preceding c_0 and a binary string after c_0 to form a codeword. This can happen only when the codeword c_0 is a substring of some other codeword.

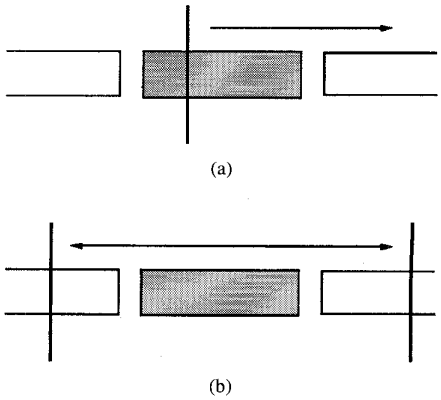

Fig. 1. Two ways that a codeword (shaded rectangle) can be decoded incorrectly. (a) First case. (b) Second case.

The two cases considered above motivate the following definition of an extended synchronizing codeword.

Definition 1: A codeword c_{es} of a prefix code C is an extended synchronizing codeword if it satisfies the following conditions:

- 1_{es} For every nonempty $\alpha \in X^*$ which is a suffix of some codeword of C, if $c_{es} = \alpha\beta$ and $\alpha \neq c_{es}$, then $\beta = \gamma\delta$, where γ is empty or a sequence of codewords, and δ is not empty, is not a prefix of any other codeword, and no codeword is a prefix of δ.
- 2_{es} c_{es} is not a substring of any other codeword.

It can be seen that the definition of an extended synchronizing codeword given above prevents the two possibilities of decoding the ESC incorrectly. Condition 2_{es} guarantees that the second possibility does not occur. Condition 1_{es} guarantees that if c_{es} is divided into two parts, and if the first part is a suffix of some other codeword, then the remaining part cannot be decoded since it is neither a prefix of any other codeword nor a sequence of codewords followed by a prefix of a codeword. In both cases, the decoding algorithm recognizes that it has received the ESC even though there can be errors preceding the ESC. Then by simply finding the first downstream position after the ESC where decoding can continue, the decoder can establish decoding synchronization. Furthermore, if the ESC is used at regular positions in the data stream, then symbol synchronization can also be established.

Of course, it is possible that an error can turn a portion of the bit stream into an ESC, or that an error can occur in the ESC. In these cases, decoding synchronization can still be established by the next downstream ESC (although symbol synchronization can be lost). The possibility of the occurrence of the above cases can be greatly reduced if a special fixed-length codeword is inserted after the ESC. The fixed-length codeword acts as a counter of the occurrence of the ESC, and is incremented after every ESC. The decoder checks the counter codeword with its own local counter, and if there is a mismatch between the two numbers, the decoder knows immediately one of the above erroneous cases has occurred. It then takes the appropriate procedure to re-establish both symbol and codeword synchronization. However, we do not address these issues in this correspondence, and assume throughout that the noise does not corrupt the ESC. An example of an extended synchronizing codeword is shown in Fig 2.

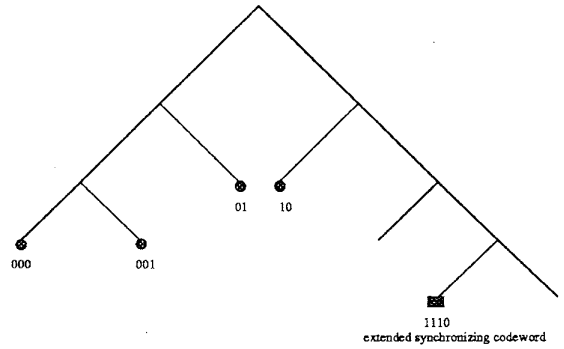

Fig. 2. An example of a code with an extended synchronizing codeword.

We now show two basic properties of extended synchronizing codewords.

Property 1: An extended synchronizing codeword $c_{es} \in C$ is not a substring of any sequence of codewords formed by codewords of $C - \{c_{es}\}$.

Proof: First, condition 2_{es} implies that an extended synchronizing codeword cannot be a substring of a single codeword from $C - \{c_{es}\}$. Hence, the only way that c_{es} could be a substring of a sequence of codewords is if $c_{es} = \alpha\gamma\delta$ where $\alpha \in \text{suffix}(C)$, γ can either be empty or a sequence of codewords in C, and $\delta \in \text{prefix}C$. However, this violates condition 1_{es} of the extended synchronizing codeword since condition 1_{es} requires that δ not be a prefix of any other codeword. Thus an ESC is not a substring of any sequence of codewords from $C - \{c_{es}\}$. □

This property is reminiscent of the notion of a clear codeword defined by Lei and Sun in [6]. Namely, they defined a clear codeword as a codeword that cannot be formed by any concatenation of other codewords. Property 1 shows that an extended synchronizing codeword satisfies such a condition. This is also the basic idea of a unique codeword discussed by Hunter and Robinson in [5].

Property 2: If a source S is designed to have a prefix code C that has at least one extended synchronizing codeword, then $E(C) > E(H)$ where H is the Huffman code for S. In other words, C cannot achieve the minimum average codeword length.

Proof: Decompose c_{es} into $\alpha\beta$ where α is either 0 or 1. By condition 1_{es}, β consists of $\gamma\delta$ where γ is a sequence of codewords and δ is not empty, is not a prefix of any other codeword, and no other codeword is a prefix of δ. Since no other codeword is a prefix of δ, and δ is not a prefix of any other codeword, then δ is a terminal node that is not used in the code tree of C. This implies the code tree of C has at least one terminal node which is not used. Thus C cannot achieve minimum average codeword length. □

Note that Property 2 of extended synchronizing codewords contrasts with synchronizing codewords. Namely, there are sources that have codes with at least one SC that achieve minimum average codeword length [3]. However, Property 2 shows that this is not possible using codes with an ESC.

IV. Upper Bounds on Overhead in Designing Codes with an ESC

In this section, we present some upper bounds on the overhead necessary in designing a binary prefix code with an extended synchronizing codeword.

Theorem 1: Let the probability distribution of a source $S = (s_1, \cdots, s_n)$ be $(p_1, \cdots, p_n)$ where $p_i \geq p_{i+1}$. Let H be the Huffman code for S with average codeword length $E(H)$. Let l_i be the length of the Huffman codeword associated with s_i (probability p_i). Then there exists a prefix code C with at least one extended synchronizing codeword and average codeword length $E(C) = E(H) + p_n l_n$.

Proof: We prove the theorem by constructing a code C with at least one extended synchronizing codeword and $E(C) = E(H) + p_n l_n$. We assume that if $p_i = p_{i+k}$ for some k, then the Huffman codewords are assigned in such a way that $l_i \leq l_{i+k}$. Using this assumption and the Huffman property, then $l_i \leq l_{i+1}$ so that l_n is the longest codeword length and is associated with the least probable symbol s_n (probability p_n). We arrange the codewords in the Huffman tree such that the codeword (length l_n) associated with s_n is 1^{l_n} (a codeword with l_n 1's). This can be done easily by relabeling the tree branch of the original Huffman codeword for s_n at the positions which are different from 1^{l_n}. By extending 1^{l_n} to $1^{2l_n-1}0$ (length $2l_n$), we claim that the codeword $1^{2l_n-1}0$ is an extended synchronizing codeword and the increase in average codeword length is $p_n l_n$. $1^{2l_n-1}0$ satisfies condition 2_{es} since it is the longest codeword of S; therefore, it cannot be a substring of any other codeword. All suffixes of codewords in C consisting entirely of 1's can have length ranging only from 0 to $l_n - 1$. Thus if the codeword $1^{2l_n-1}0$ is decomposed into two parts, with the first part being a suffix of some other codeword, then the second part will be a binary string of the form 1^k0 where k ranges from l_n to $2l_n - 1$. This binary string is not a prefix of any other codeword so that condition 1_{es} is satisfied. Hence, $1^{2l_n-1}0$ is an extended synchronizing codeword. □

Corollary 1: Given the same situation as in Theorem 1, a source S admits a prefix code C with at least one extended synchronizing codeword with probability p_i such that the average codeword length of C is

$$E(C) = E(H) + p_i l_n + (p_i - p_n)(l_n - l_i).$$

Proof: Using the same procedure as in the proof of Theorem 1, we construct the extended synchronizing codeword with probability p_n and length $2l_n$. Then we just exchange the symbol s_i and s_n, thus s_i takes the codeword of length $2l_n$ and s_n takes the codeword of length l_i. The increase in the average codeword length is $2p_i l_n + p_n l_i - p_i l_i - p_n l_n$ which is equal to $p_i l_n + (p_i - p_n)(l_n - l_i)$. □

When $i = n$, from Corollary 1 we get $E(C) = E(H) + p_n l_n$ which reduces to the equality in Theorem 1. If $i < n$, then $p_i \geq p_n$ and $l_n \geq l_i$, thus

$$(p_i - p_n)(l_n - l_i) \geq 0.$$

So the additional overhead is at least $p_i l_n$ which is strictly greater than zero. The reason that one might want to design an ESC to have a certain probability p_i is that the ESC may be put at regular positions in the bit stream to be used as a marker, to stop error propagation. The frequency with which one desires to use the marker, is related to p_i.

Theorem 2: If the depth of a maximal complete subtree in a Huffman code is d, then there exists a prefix code C with an extended synchronizing codeword of probability p_n, and

$$E(C) = E(H) + p_n(d+1) \leq E(H) + \frac{1}{n}(\log_2 n + 1).$$

Proof: For each codeword c, let

$$s_1(c) = \max\{k | 1^k \in \text{suffix}(c)\}$$

That is, $s_1(c)$ is the number of 1's at the end of the codeword c.

Rotate the original Huffman tree as necessary in order to get the longest codeword c_n to be all 1's. Now, if $s_1(c) \leq d$ for all $c \neq c_n$ then we can easily construct a prefix code with the desired properties by appending the sequence 1^d0 to the codeword for c_n. In this case, c_n is easily seen to be an ESC, and $E(C) = E(H) + p_n(d+1)$ since the only difference from the original code H is that the length of c_n has been increased by $d+1$. To complete the proof, we will argue that by suitable rotations about various nodes of the tree we can always arrange the tree so that $s_1(c) \leq d$ for all $c \neq c_n$.

Consider the original Huffman tree H. Mark the root node and all nodes in the path from the root node to c_n. Repeat the following until there are no more unmarked nodes. Visit the unmarked node of least depth—i.e., closest to the original root node. (If there is more than one of minimal depth then just pick one arbitrarily). When we visit a node N, let c_N be the codeword with prefix corresponding to N followed by all 1's. That is, c_N is in the subtree with root node N and is the codeword obtained by always following the branch labeled 1 after the node N. We will either leave the tree unchanged and simply mark all nodes in the path from N to c_N (i.e., nodes of the form $N\,1^j$ for some $j \geq 0$), or else we will rotate various parts of the subtree with root node N (leaving the rest of the tree alone) and afterwards mark all the nodes in the path from N to the (new) codeword c_N.

Our rule for rotating the subtree with root node N is as follows. If $s_1(c_N) \leq d$ then leave the tree alone, mark the appropriate nodes, and proceed to consider the next unmarked node. If $s_1(c_N) > d$ then rotate the subtree as necessary (about node N and any other nodes in the subtree) to ensure that for the rotated tree $s_1(c_N) \leq d$. This can always be done for the following reasons. By construction, the path leading to node N must end with a 0, otherwise node N would have already been marked when considering some previous node. Therefore, $s_1(c_N)$ is simply the depth of the subtree starting at node N and following paths labeled 1 all the way to a leaf node. But since the depth of the maximal complete subtree of the original tree H is d, the subtree of every node in H (and so in particular the subtree with root node N) has some leaf node of depth d (with respect to the subtree). Hence, we need only rotate about the nodes in the subtree so that the leaf node of depth at most d in the subtree corresponds to all 1's.

In the above procedure, each time we visit a new node we mark at least one node (namely, the node visited). Since there are only a finite number of nodes in the original tree H, the procedure will eventually terminate. Also, note that once a node N has been visited (and the subtree has been rotated as necessary) then all nodes in the path from N to c_N will be left alone since by construction they will be marked and not visited and moreover they will not be in the subtree of any unmarked node. Therefore, no downstream adjustments will alter the fact that $s_1(c_N) \leq d$. Thus after all the nodes have been visited, the resulting tree will satisfy $s_1(c) \leq d$ for all $c \neq c_n$. □

Using results from Capocelli *et al.* [2] and Berger and Yeung [1], a source S admits a prefix code C with an ESC of probability p_n and $E(C) = E(H) + \frac{1}{2}$. But for all $n \geq 6$, $p_n(d+1)$ is a tighter bound than $\frac{1}{2}$, and the two bounds are equal only when $n = 6$ and $n = 8$. In addition, $p_n(d+1) \to 0$ as $n \to \infty$, which implies that the overhead for designing a binary prefix codes with an extended synchronizing codeword becomes negligible as the number of source symbols increase.

V. Conclusion

This correspondence introduced the notion of extended synchronizing codewords which can be inserted into a bit stream to prevent error propagation. The ESC's can serve as partition markers to allow both symbol and codeword synchronization even in the presence of errors. We derived certain essential properties of ESC's, and proved some upper bounds on the amount of overhead in designing a binary prefix code with at least one ESC.

Acknowledgment

The authors wish to thank the anonymous referees for helpful comments.

References

[1] T. Berger and R. W. Yeung, "Optimum 1-ended binary prefix codes," *IEEE Trans. Inform. Theory*, vol. 36, no. 6, pp. 1435–1441, Nov. 1990.

[2] R. M. Capocelli, A. A. D. Santis, L. Gargano, and U. Vaccaro, "On the construction of statistically synchronizable codes," *IEEE Trans. Inform. Theory*, vol. 38, no. 2, pp. 407–414, Mar. 1992.

[3] T. J. Ferguson and J. H. Rabinowitz, "Self-synchronizing Huffman codes," *IEEE Trans. Inform. Theory*, vol. IT-30, no. 4, pp. 687–693, July 1984.

[4] B. L. Montgomery and J. Abrahams, "Synchronization of binary source codes," *IEEE Trans. Inform. Theory*, vol. IT-32, no. 6, pp. 849–854, Nov. 1986.

[5] R. Hunter and A. H. Robinson, "International digital facsimile coding standards," *Proc. IEEE*, pp. 854–867, July 1980.

[6] S.-M. Lei and M.-T. Sun, "An entropy coding system for digital HDTV applications," *IEEE Trans. Circuits Syst. for Video Technol.*, vol. 1, no. 1, pp. 147–154, Mar. 1991.

[7] W.-M. Lam and A. R. Reibman, "Self-synchronizing variable-length codes for image transmission," in *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, Mar. 1992, vol. 3, pp. 477–480.

[8] Y. Takishima, M. Wada, and H. Murakami, "Error states and synchronization recovery for variable length codes," *IEEE Trans. Commun.*, vol. 42, no. 2–4, pp. 783–792, Feb./Mar./Apr. 1994.