Bidirectionally Decodable Streams of Prefix Code-Words

Bernd Girod, *Fellow, IEEE*

Abstract— **A new general scheme is introduced that allows bidirectional decoding of variable length coded bitstreams from either end. Except for a small fixed number of extra bits appended to a sequence of code words, the scheme is as efficient as Huffman coding. The extra operations required at coder and decoder are code word reversal and one EXOR for each bit.**

Index Terms—**Affix codes, biprefix codes, error resilient source coding, Huffman coding, reversible variable length coding.**

I. INTRODUCTION

When variable length codes (VLC's) are utilized for source compression, bit errors resulting from transmission over a noisy communication link can have catastrophic consequences. One erroneously decoded symbol often leads to a loss of synchronization, and many, or possibly all following decoded symbols are faulty. In practice, one inserts unique synchronization patterns into the VLC bitstream in regular intervals, such that synchronization can be regained. We refer to the VLC payload bits between two synchronization patterns as one *frame* in this letter.

For a given set of symbol probabilities, a minimum redundancy code can be constructed using Huffman's algorithm [1]. The Huffman code tree is usually mapped into a prefix code, such that no code word is the prefix of another code word. Thus, code-words can be concatenated and are uniquely decodable without extra separation symbols. Parsing the bitstream in forward direction (i.e., beginning with the start of a frame), we can decode the bitstream one symbol at a time. In general, a concatenation of prefix code-words is not decodable in reverse direction, i.e., parsing a frame from its end, we can uniquely determine the symbol separations only with considerable delay and computational effort [2]. For some symbol strings, it might be required to process the bitstream all the way from the end to the beginning of the frame, until following symbols can be uniquely decoded. Therefore, a bit error occuring somewhere inside a frame renders the remainder of the frame useless. In such situations, it is highly desirable to have a VLC that can also be decoded symbol by symbol in reverse direction, such that the remainder of the frame can be recovered and only few symbols are affected.

Given the importance of reverse decoding for practical systems operating over error-prone channels, amazingly little research has been published that addresses the problem [2]–[4]. Only recently, there has been an increasing interest in the context of error resilient video coding methods, notably for the MPEG-4 standardization and for mobile extensions of ITU-T Recommendation H.263 [5]–[7]. The approach to reversible variable length coding (RVLC) has been in general to design a code table that possesses both the prefix and the suffix condition, i.e., each code word is neither the prefix nor the suffix of any other code word. The bitstream resulting from the concatenation of such *biprefix* (or *affix* [2]) code-words is then decodable in either direction. Unfortunately, only certain optimal prefix codes, such as the Golomb–Rice codes or the Exp-Golomb codes, can be mapped into an equally efficient biprefix code [5], [6]. For general symbol probabilities the biprefix constraint often leads to codes that are less efficient than Huffman codes [2], [4].

In this letter, a new general method is presented that generates a bidirectionally decodable bitstream composed of VL prefix code words. Rather than designing a severely constrained biprefix code, we modify the bitstream resulting from the concatenation of prefix code words (e.g., Huffman code words) appropriately, such that reversible decoding becomes possible. With the exception of a small fixed number of extra bits appended to the end of a frame, the scheme is as efficient as Huffman coding.

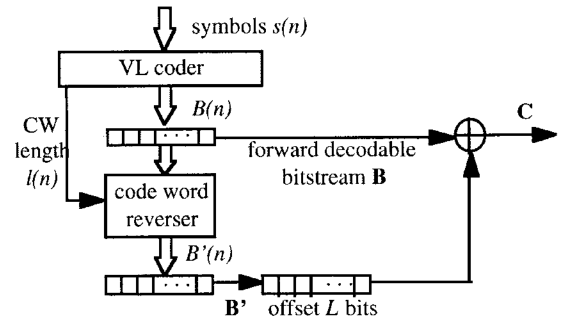

Fig. 1. Blockdiagram of the reversible variable length coder.

II. NEW METHOD

The new method is based on the idea to combine prefix and suffix code words representing the same symbol string at the coder, and to convert this bitstream into a concatenation of appropriately oriented prefix code words in the decoder. We first show the coder, then the decoder operating in forward and in reverse direction.

Manuscript received June 17, 1998. The associate editor coordinating the review of this letter and approving it for publication was Prof. Y. Bar-Ness.

The author is with the Telecommunications Laboratory, University ofErlangen-Nuremberg, 91058 Erlangen, Germany (e-mail: girod@nt.e-technik.uni-erlangen.de).

Publisher Item Identifier S 1089-7798(99)07356-1.

A. Encoder

Fig. 1 shows the blockdiagram of the encoder. A string of symbols $s(n), n = 1, \cdots, N$ is encoded with a prefix code that represents the nth symbol $s(n)$ by $l(n)$ bits $B(n) = b_1(n)|b_2(n)|b_3(n)|\cdots|b_{l(n)}(n)$. The vertical bar $a|b$ represents the concatenation of bit patterns a and b. In a table-based implementation, one can simply store the code word length l together with $b_1|b_2|b_3|\cdots|b_l$ for each possible symbol s. In a conventional variable length coder, one would now concatenate the variable length (VL) code-words by shifting them serially out of the register producing the bitstream

$$\begin{aligned}\boldsymbol{B} &= \boldsymbol{B}(\mathbf{1})|\boldsymbol{B}(\mathbf{2})|\cdots|\boldsymbol{B}(\boldsymbol{N}) \\ &= b_1(1)|b_2(1)|b_3(1)|\cdots|b_1(n)|b_2(n)| \\ &\quad \cdots|b_{l(N)-1}(N)|b_{l(N)}(N). \end{aligned} \tag{1}$$

In our new coder, we reverse each individual code word, transforming $B(n)$ into $B'(n)$, such that

$$B'(n) = b_{l(n)}(n)|b_{l(n)-1}(n)|\cdots|b_2(n)|b_1(n). \tag{2}$$

The bitstream of concatenated reversed code words

$$\boldsymbol{B}' = \boldsymbol{B}'(\mathbf{1})|\boldsymbol{B}'(\mathbf{2})|\cdots|\boldsymbol{B}'(\boldsymbol{N}) \tag{3}$$

possesses the suffix condition, i.e., it can be decoded, one symbol at a time, in reverse, but not in forward direction. We generate a bidirectionally decodable bitstream by combining

$$\begin{aligned}\boldsymbol{C} = &\Big(\boldsymbol{B}(\mathbf{1})|\boldsymbol{B}(\mathbf{2})|\cdots|\boldsymbol{B}(\boldsymbol{N})|\underbrace{0|\cdots|0}_{\boldsymbol{L}\text{ zeros}}\Big) \\ &\oplus\Big(\underbrace{0|\cdots|0}_{L\text{ zeros}}|B'(1)|B'(2)|\cdots|B'(N)\Big)\end{aligned} \tag{4}$$

where $\oplus$ represents a bitwise exclusive-or (EXOR) operation. Thus, the forward decodable bitstream $\boldsymbol{B}$ is appended with L trailing zero bits, while the reversely decodable bitstream $\boldsymbol{B}'$ is offset (delayed) by L bits and augmented by L leading zeros. For bidirectional decodability, the offset has to obey

$$L \geq l_{\max} = \max_n \; l(n) \tag{5}$$

as will be apparent when considering the decoding procedure in the next section. In practice, one simply chooses a fixed L equal to the maximum code word length $l_{\max}$ of the VL code, which minimizes redundancy without requiring knowledge about the actually occuring code word lengths $l(n)$ for a frame.

Note that an extension of the scheme to nonbinary VLC is straightforward. For an N-ary code, one simply replaces the exclusive-or operation by an addition modulo N.

B. Decoding in Forward Direction

Fig. 2 shows the blockdiagram of the decoder. The forward decodable bitstream is recovered by

$$\begin{aligned}&B(1)|B(2)|\cdots|B(N)|\underbrace{0|\ldots|0}_{L\text{ zeros}} \\ &= \boldsymbol{C} \oplus \Big(\underbrace{0|\cdots|0}_{L\text{ zeros}}|\boldsymbol{B}'(\mathbf{1})|\boldsymbol{B}'(\mathbf{2})|\cdots|\boldsymbol{B}'(\boldsymbol{N})\Big)\end{aligned} \tag{6}$$

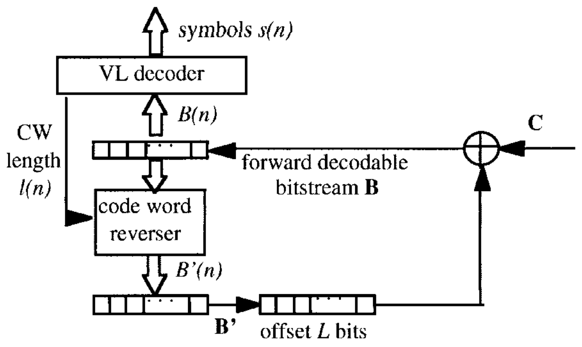

Fig. 2. Blockdiagram of the reversible variable length decoder.

i.e., the bitstream $\boldsymbol{C}$ is combined in an EXOR operation with the bitstream of reversed VL code words, offset by L bits. The bitstream $B(1)|B(2)|\cdots$ can be decoded symbol by symbol. Each time the VL decoder has recovered another symbol $s(n)$, it signals the length $l(n)$ to the code-word reverser, and the corresponding bits $B(n)$ are reversed to produce $B'(n)$. With an offset of L bits, $B'(n)$ is fed back and EXORed with the next $l(n)$ bits of $\boldsymbol{C}$ to produce the next $l(n)$ bits of the forward decodable bitstream. Thus $l(n)$ bits are "consumed" from the forward decodable bitstream, but $l(n)$ new bits of $\boldsymbol{B}$ are recovered after code word reversal.

The offset L has to be at least the maximum code-word length $l_{\max}$, to ensure that there is always a sufficient number of bits in the forward decodable bitstream $\boldsymbol{B}$ to decode the next symbol. If the maximum length code word is encountered, all bits available in the forward decodable bitstream at that time will be "consumed" by the VL decoder to decode this symbol, if $L = l_{\max}$.

As apparent from (6), the last L bits of $\boldsymbol{C}$ are redundant for decoding in forward direction. Since the last L bits of $\boldsymbol{B}'$ are identical with the last L bits of $\boldsymbol{C}$, the forward decodable bitstream ends with L trailing zeros. These can be used to check synchronization. If the trailing L bits do not decode to zero, this indicates that synchronization was lost at the end of the frame due to bit errors.

C. Decoding in Reverse Direction

Any bitstream $\boldsymbol{C}$ can be decoded in reverse direction by simply feeding it backward into the decoder in Fig. 2. The last L bits of $\boldsymbol{C}$ are identical with the last L bits of $\boldsymbol{B}'$ (4). Since the concatenated code-words of $\boldsymbol{B}'$ are reversed prefix code-words, i.e., suffix code-words, we can easily decode the last symbol $s(N)$ from $B'(N)$. Once we have decoded $s(N)$, we also know the code-word length $l(N)$, and can thus reverse $B'(N)$ to produce $B(N)$. By EXORing the shifted $B(N)$ with the appropriate bits of $\boldsymbol{C}$, one recovers $l(N)$ bits of $\boldsymbol{B}'$ and the next symbol $s(N-1)$ can be decoded. The process is repeated until the beginning of frame $\boldsymbol{C}$ is reached or an error is detected.

Again, the lastly recovered L bits leading the reversely decodable bitstream $\boldsymbol{B}'$ must be zero. Nonzero bits indicate a loss of synchronization at the end of the decoding process due to bit errors.

III. Concluding Remarks

A new scheme has been introduced that allows bidirectional decoding of VLC bitstreams from either end. We modify the bitstream resulting from the concatenation of prefix code-words (e.g., Huffman code-words) appropriately, such that reversible decoding becomes possible. Except for a small fixed number of extra bits appended to the end of a frame, the scheme is as efficient as Huffman coding.

Compared to conventional VL coding, the extra operations required at coder and decoder are code word reversal and one EXOR for each bit. These can be cascaded with a conventional VL coder and decoder, thus allowing a straightforward extension of existing VL coders, or the optional use of bidirectionally decodable bitstreams with little extra effort.

To mitigate the effects of bit errors through combined forward and backward decoding of VLC bitstreams, reliable error detection is required. Error detection is not part of this scheme and has to be carried out by an appropriate combination of methods. For example, extra parity check bits could be inserted to roughly locate an error. Often, this is not even neccessary, since bit errors in VLC bitstreams tend to propagate with catastrophic consequences. It is therefore easy to detect a garbled video, audio, or text stream, e.g., from syntax violations, and thus locate an error.

The L extra bits needed for bidirectional decodability also provide a certain error resilience. In particular, up to L consecutively erased bits can be tolerated without information loss. The precise number depends on the exact alignment of the erasure burst relative to the code word boundaries. An erasure of $L - l_{\max} + 1$ consecutive bits can always be corrected, if no other bit error occurs in the frame. Hence, the burst erasure resilience can be increased easily by increasing L. Note, however, that with multiple bit erasures that are further apart the entire segment in between them cannot be recovered, and other techniques such as self-synchronizing variable length codes [8], [9] or variable-to-fixed length *Tunstall* coding [10] might be appropriate.

Finally, bidirectionally decodable VLC bitstreams are also useful for other applications, e.g., random access in compressed data bases. Random access to information at the end of a frame is faster with reverse decoding.

References

[1] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proc. IRE*, vol. 40, p. 1098, Sept. 1952.

[2] A. S. Fraenkel and S. T. Klein, “Bidirectional Huffman coding,” *Computer J.*, vol. 33, no. 4, pp. 297–307, 1990.

[3] J. Berstel and D. Perrin, *Theory of Codes*. Orlando, FL: Academic, 1985.

[4] Y. Takashima, M. Wada, and H. Murakami, “Reversible variable length codes,” *IEEE Trans. Commun.*, vol. 43, pp. 158–162, Feb./Mar./Apr. 1995.

[5] J. Wen and J. D. Villasenor, “A class of reversible variable length codes for robust image and video coding,” in *Proc. 1997 IEEE Int. Conf. on Image Processing, ICIP-97*, Santa Barbara, CA, vol. 2, pp. 65–68, Oct. 1997.

[6] ——, “Reversible variable length codes for efficient and robust image and video coding,” in *Proc. IEEE 1998 Data Compression Conf.*, Snowbird, UT, Mar. 1998.

[7] R. Talluri, “Error resilient video coding the ISO MPEG-4 standard,” *IEEE Communications Mag.*, vol. 36, pp. 112–119, June 1998.

[8] P. G. Neumann, “Efficient error-limiting variable length codes,” *IRE Trans. Inform. Theory*, vol. IT-8, pp. 292–304, July 1962.

[9] T. J. Ferguson and J. H. Rabinowitz, “Self-synchronizing Huffman codes,” *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 687–693, July 1984.

[10] B. P. Tunstall, “Synthesis of noiseless compression codes,” Ph.D. dissertation, Georgia Inst. Technol., Atlanta, Sept. 1967.