DESING OF REVERSIBLE VARIABLE-LENGTH CODES USING PROPERTIES OF THE HUFFMAN CODE AND AVERAGE LENGTH FUNCTION

*Wook-Hyun Jeong, **Young-Suk Yoon, and ** Yo-Sung Ho

*Samsung Electronics Co., LTD
416 Maetan3-dong, Paldal-gu, Suwon-si, Gyeonggi-do, Korea

**Gwangju Institute of Science and Technology (GIST)
1 Oryong-dong, Buk-gu, Gwangju, 500-712, Korea
*wh75.jeong@samsung.com, **{ysyoon, hoyo}@gist.ac.kr

ABSTRACT

Variable-length codes (VLCs) are generally employed to improve compression efficiency using data statistics. However, VLCs are very sensitive to bit errors in noisy transmission environments, such as mobile channels. Recently, several reversible variable-length codes (RVLCs) have been introduced due to recovering information from corrupted compressed bitstreams and enhancing robustness of VLCs to bit errors. However, existing RVLCs have some rooms for improvement in coding efficiency. In this paper, we propose a new design algorithm for efficient symmetrical and asymmetrical RVLCs by employing essential information from the Huffman code and the property of the average length function. The proposed algorithm has demonstrated improved coding efficiency over existing RVLC algorithms.

1. INTRODUCTION

Although most image and video coding standards include variable-length codes (VLCs), such as the Huffman code [1] and the arithmetic code [2], they are so sensitive to bit errors that their decoders may lose synchronization, which may lead to loss of several video frames.

In recent years, reversible variable-length codes (RVLCs) have been introduced as one of the error resilience tools to reduce this problem. In existing VLCs with a resynchronization marker, we have to throw out all the received data until the next resynchronization marker after even a single bit error. However, in RVLC with a resynchronization marker, we can decode the bitstream both in the forward and backward directions and recover uncorrupted video data as much as possible from the received erroneous bitstream.

RVLC algorithms can be categorized into two different classes: symmetrical and asymmetrical RVLCs according to their bit patterns. While MPEG-4 includes an asymmetrical RVLC [3], H.263+ employs a symmetrical RVLC [4].

Takishima *et al*. [5] proposed the first work which specified the method for constructing symmetrical and asymmetrical RVLCs based on a given Huffman code to make their average codeword lengths close to that of the optimal Huffman code. Tsai *et al*. [6] improved this algorithm and reduced the average codeword length. Recently, Tseng *et al*. [7] introduced a non-Huffman-code-based scheme for symmetrical RVLC. This is an exhaustive algorithm with a bounding function and backtracking schemes. Lin *et al*. [8] extended Tseng's method to the asymmetrical RVLC.

Analysis of these RVLC algorithms, however, shows some rooms for improvement with respect to coding efficiency. In both Takishima's and Tsai's algorithms, some restrictions are imposed on the construction process. Consequently, they may miss some efficient codewords. Although Tseng's and Lin's algorithms provide better performance than conventional Huffman-code-based algorithms, they are not clear in many critical factors to design RVLCs, such as the starting bit length and codeword selection mechanisms. Besides, since Tseng and Lin assumed that the sum of local optimizations can lead to the global optimization, which is not always true, naturally their proposed backtracking algorithms are limited.

In this paper, we propose new code design algorithms for symmetrical and asymmetrical RVLCs using properties of the Huffman code and the average length function. In order to obtain more efficient RVLCs, we adopt critical information from the Huffman code. Moreover, we define and exploit the average length function that is useful to search for efficient reversible codewords at each level of the proposed algorithms.

2. PROPOSED RVLC ALGORITHM

2.1. Property of the Huffman Code

We should determine several elements that organize a target RVLC, such as the shortest bit length in the RVLC, the number of codewords at each level, and corresponding codewords. If we apply the codeword assignment of the given Huffman code, we can concrete the region, where the more efficient RVLC exists, adapting the critical elements of RVLCs.

In the optimal Huffman code, we assign the shortest codeword to the most probable symbol, whose bit length is determined by the source distribution and the number of

0-7803-8554-3/04/$20.00 ©2004 IEEE.

symbols. However, the bit length L_{min} of the shortest Huffman codeword is critical to build the optimal code under the given distribution.

Since asymmetrical RVLCs do not concern bit-patterns of codewords, they are close to the Huffman code. Thus, we adopt a critical description L_{min} of a given Huffman code as the starting bit level of an asymmetrical RVLC. However, for the source with a nearly uniform distribution, results with L_{min}–1 rather than L_{min} show better performance for the symmetrical RVLC. On the other hand, for the input data with a highly skewed distribution, L_{min}+1 is a good empirical choice. Therefore, in symmetrical RVLCs, we can take one of the three values, L_{min}–1, L_{min} , and L_{min}+1, as the starting bit level based on the given source distribution.

Let the bit length vector $n(i)$ denote the number of codewords with the bit length i, and $n_{Huff}(i)$ and $n_{RVLC}(i)$ be bit length vectors of the Huffman code and RVLC, respectively. Since $n_{RVLC}(i)$ is usually smaller than $n_{Huff}(i)$ at lower levels due to the suffix condition, $n_{RVLC}(L_{min})$ should be larger than or equal to $n_{Huff}(L_{min})$ to increase the priority of the highest level. The maximum number of codewords at level L_{min} is 2^{Lmin} and the range of is given by:

$$n_{Huff}(L_{min}) \le n_{RVLC}(L_{min}) \le 2^{L_{min}} \tag{1}$$

In addition, the choice of $\mathbf{Z_L}$ at the starting level is useful [9]. $\mathbf{Z_L}$ is composed of only L '0' bits and it should be selected at least once, along with the codeword consisting of all '1' bits, over the whole bit levels. From the choice of $\mathbf{Z_L}$, we can provide at least one codeword at the highest level. In the asymmetrical RVLC, the bit length of $\mathbf{Z_L}$ is L_{min} while we should employ one value in $\mathbf{Z_L}$s for L_{min}–1, L_{min}, and L_{min}+1.

The Huffman code for highly skewed sources does not assign any codewords to a few bit levels and we define these levels as the skipped level L_{skip}. Moreover, we separate whole bit levels into two groups, upper levels and lower levels, where the upper levels contain a half of the number of symbols. In order to obtain more efficient RVLCs, we should assign more codewords to shorter bit levels in the upper levels. Simultaneously, we have to guarantee lower levels more available candidates. Obviously, this strategy is contractive. We figure out this contraction using the level-skipping adaptation. In the level-skipping adaptation, we do not assign any reversible codewords to L_{skip}.

2.2. The Average Length Function

As a measure of coding performance, we use the average codeword length $\overline{L}(X)$ of a VLC of the source X, which is a convex function (∪) [10].

We define a set $R(L;M)$ whose components are all RVLCs that have the same reversible codewords in all the upper levels over level L and $n_{RVLC}(L) = M$.

$$R(L;M) = \{RVLC \mid \Psi_{L_{min}}^{L-1} \text{ and } n_{RVLC}(L) = M\} \tag{2}$$

where Ψ_m^n is a set whose components are already chosen from all reversible codewords of bit length from m to n for $m \le n$.

In addition, we define the average length function $f_{\overline{L}}$, which is the function of $n_{RVLC}(L)$ and whose range is the minimum value of the average lengths in the set R for the given source distribution.

$$f_{\overline{L}} : n_{RVLC}(L) \to \min(\overline{L}(R)) \tag{3}$$

where $\min(\overline{L}(R))$ is the minimum value of average codeword lengths of all components in the set $R(L;M)$ for the given source distribution.

From the average length function, we can select more efficient reversible codewords at each level. The average length function $f_{\overline{L}}$ is a convex function (∪) that has the minimum value in a certain interval. We assume that this function gradually and monotonically decreases or increases near the minimum value. For $n_{RVLC}(L_{min})$ ranging from $n_{Huff}(L_{min})$ to $2^{L_{min}}$, the average length function has the minimum value in [$n_{Huff}(L_{min})$, $2^{L_{min}}$], as shown in Fig. 1. Since $f_{\overline{L}}$ is convex, the local minimum in this interval related to $R(L;M)$ can represent the global minimum value solely at level L. Thus, more efficient RVLC is located on the minimum value of $f_{\overline{L}}$. After selecting $n_{RVLC}(L)$, we include corresponding codewords at level L in the set $\Psi_{L_{min}}^{L-1}$ and update the set Ψ by $\Psi_{L_{min}}^{L}$.

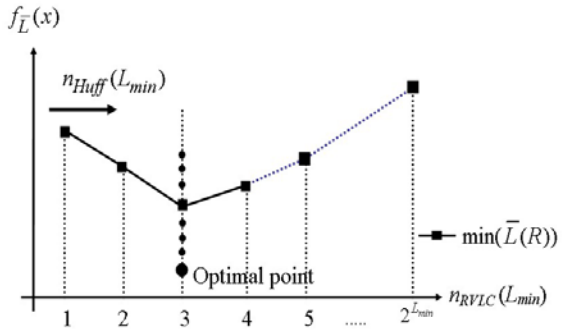

Fig. 1 Minimum value of the average length function

In symmetrical RVLCs, since we should consider three starting levels, L_{min} –1, L_{min}, and L_{min} +1, we extend the domain of $f_{\overline{L}}$. If L_{skip} is not observed in the given Huffman code, we determine the starting bit level of the RVLC after comparing $f_{\overline{L}}$s for both L_{min} –1 and L_{min}. Otherwise, we should select the starting bit level between L_{min} and L_{min} +1 in the same way. Additionally, although L_{min} +1 is skipped in the Huffman code, we assign codewords to level L_{min} +1.

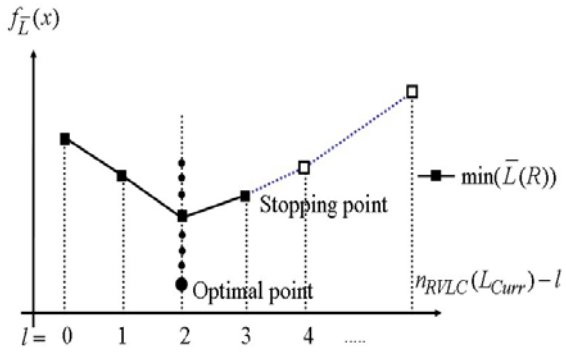

Fig. 2 The number of reversible codewords at current level

In upper levels, we select available efficient codewords at each level as follows.

1) Find all available candidates at the current level L_{curr} .
2) Generate $\begin{pmatrix} n_{RVLC}(L_{curr}) \\ n_{RVLC}(L_{curr})-l \end{pmatrix}$ RVLCs and calculate the average length function $f_{\bar{L}}$ where $\begin{pmatrix} a \\ b \end{pmatrix}$ is the number of combinations of size b from a set of size a and l is increased by 1 from 0.
3) Due to the convex property, if the current value of $f_{\bar{L}}$ is larger than that of the previous one, select the previous codeword, as shown in Fig 2.

The proposed design procedure for efficient RVLCs is

1) Determine available codewords at L_{min} for asymmetrical RVLCs applying the level-skipping adaptation and the choice of $\mathbf{Z_L}$. For symmetrical RVLCs, the starting bit level is one of the three values based on the existence of skipped levels.
2) In upper levels, search for reversible codewords at each level with the average length function.
3) In lower levels, select all available codewords satisfying both the prefix condition and the suffix condition at each level.

4. EXPERIMENTAL RESULTS

Table I lists average codeword lengths of symmetrical and asymmetrical RVLCs for file sets from Canterbury Corpus[11] designed by Tsai's[6], Tseng's[7], Lin's[8], and the proposed algorithms. Various file sets from Canterbury Corpus are used to measure and compare the compression performance.

Coding performance of the proposed symmetrical RVLC scheme is better than those of Huffman-code-based algorithms; however, we cannot improve the coding efficiency of the backtracking algorithm. On the other hand, as shown in Table I, we reduce average codeword lengths for asymmetrical RVLCs significantly over all existing methods. Overall, the average lengths of proposed asymmetrical RVLCs are about 3% shorter than those of Lin's asymmetrical RVLCs.

There are skipped bit levels in Huffman codes for F2, F4, F8, F9, F10, and F11 and we have applied the level-skipping adaptation to RVLCs for these files. In Table I, we note that the proposed $\mathbf{Z_L}$ adaptation, the level-skipping adaptation, and the average length function can improve the coding efficiency successfully. In addition, we observe that the proposed algorithm is superior to existing algorithms for the source that has a lot of symbols with highly skewed distributions.

In Table II, we list codeword assignments for the English alphabet with symmetrical RVLCs and asymmetrical RVLCs designed by the existing and proposed algorithms, and compare their coding performances in terms of the average codeword length. In the case of symmetrical RVLCs, Tseng's scheme and the proposed scheme show the same average codeword length. Coding performance has been improved significantly with the asymmetrical RVLC. The average codeword length of our RVLC is about 0.34% shorter than that of Lin's RVLC. Since any bit level is not skipped, we have not employed the level-skipping adaptation to symmetrical and asymmetrical RVLCs for English alphabets.

5. CONCLUSIONS

In this paper, we have proposed a new approach to design symmetrical and asymmetrical RVLC algorithms effectively. Since Huffman-code-based algorithms have limitations in their code design process, they may miss some available codewords. Non-Huffman-code-based algorithms assumed that the local minimum of the codeword length be the global minimum, which is not always true. In order to design more efficient RVLC algorithms for different source statistics, we use essential information from the Huffman code and the average length function. The property of the Huffman code is used for determining elementary components of RVLC algorithms. The proposed average length function search for efficient codewords at each level effectively. We have identified better coding performance of the proposed schemes with various source files and English alphabet. Experimental results demonstrate that the proposed algorithm improve the coding performance over all existing algorithms.

ACKNOWLEDGEMENT

This work was supported in part by Gwangju Institute of Science and Technology (GIST), in part by the Ministry of Information and Communication (MIC) through the Realistic Broadcasting Research Center (RBRC) at GIST.

REFERENCES

[1] D. Huffman, "A method for the construction of minimum redundancy code," *Proc. Inst. Radio Engr.,* vol. 40, pp. 1098-1101, Sept. 1952.

[2] J.J. Risannen and G.G. Langdon. Jr., "Arithmetic coding," *IBM J. Res. Develop.,* 23, pp. 149-162, 1979.

[3] ISO/IEC 14496-2, "Information technology – coding of audio/visual objects," *Final Draft International Standard,* Part 2 : Visual, Oct. 1998.

[4] ITU-T Recommendation H.263, "Video coding for low bit rate communications," Annex V, 2000.

[5] Y. Takishima, M. Wada, and H. Murakami, "Reversible variable length codes," *IEEE Trans. Communications,* vol. 43, pp. 158-162, Feb. 1995.

[6] C.W. Tsai and J.L. Wu, "A modified symmetrical reversible variable length code and its theoretical bounds," *IEEE Trans. Information Theory,* vol. 47, pp. 2543-2548, Sept. 2001.

[7] H.W. Tseng and C.C. Chang, "Construction of symmetrical reversible variable length codes using backtracking," *Computer Journal,* vol. 46, no. 1, Jan. 2003.

[8] C.W. Lin, Y.J. Chuang, and J.L. Wu, "Generic construction algorithm for symmetric and asymmetric RVLCs," *Proc. IEEE International Conference on Communication Systems,* vol. 2, pp. 968-972, Nov. 2002.

[9] W.H. Jeong and Y.S. Ho, "A new construction algorithm for symmetrical reversible variable-length codes from the Huffman code," *Lecture Notes in Computer Science 2869,* pp. 675-682, Nov. 2003.

[10] T.M. Cover and J.A. Thomas, *Elements of Information Theory,* Wiley, New York, 1991.

[11] http://corpus.canterbury.ac.nz

Table I. Coding performances of symmetrical and asymmetrical RVLCs for various source file sets

Files		No. of symbols	Huffman code	Symmetrical RVLC			Asymmetrical RVLC		
				Tsai's method	Tseng's method	Proposed method	Tsai's method	Lin's method	Proposed method
			Average length	Average length	Average length	Average length	Average length	Average length	Average length
F1	asyoulik.txt	68	**4.84465**	5.27886	5.21025	**5.21025**	5.01142	5.00954	**4.85262**
F2	Alice29.txt	74	**4.61244**	5.01398	4.93155	**4.93155**	4.80326	4.68871	**4.68856**
F3	Xargs.1	74	**4.92382**	5.39863	5.33996	**5.33996**	5.07334	5.16087	**4.93577**
F4	grammar.lsp	76	**4.66434**	5.04757	5.01774	**5.01774**	4.85461	4.78581	**4.72571**
F5	Plabn12.txt	81	**4.57534**	4.94473	4.89527	**4.89527**	4.80659	4.64910	**4.63477**
F6	lcet10.txt	84	**4.69712**	5.12232	5.01682	**5.01682**	4.87868	4.74177	**4.72741**
F7	cp.html	86	**5.26716**	5.85839	5.81173	**5.81173**	5.37113	5.77080	**5.28112**
F8	Fields.c	90	**5.04090**	5.47596	5.46332	**5.46332**	5.26987	5.20278	**5.08843**
F9	Ptt5	159	**1.66091**	1.77735	1.75992	**1.75992**	1.71814	1.70401	**1.67630**
F10	Sum	255	**5.36504**	6.10683	6.03917	**6.03917**	5.49767	6.01870	**5.41683**
F11	kennedy.xls	256	**3.59337**	4.25681	4.27209	**4.27209**	3.89401	3.85384	**3.78413**

Table II. Coding performances of symmetrical and asymmetrical RVLCs for the English alphabet

Occurrence Probability		Huffman code		Symmetrical RVLC						Asymmetrical RVLC					
				Tsai's algorithm		Tseng's algorithm		Proposed algorithm		Tsai's algorithm		Lin's algorithm		Proposed algorithm	
		L	codeword	L	codeword	L	codeword	L	codeword	L	codeword	L	codeword	L	codeword
E	0.14878570	3	001	3	010	3	000	3	000 ($\mathbf{Z}_3$)	3	000	3	000	3	000 ($\mathbf{Z}_3$)
T	0.09354149	3	110	3	101	3	111	3	111	3	111	3	100	3	101
A	0.08833733	4	0000	4	0110	3	010	3	010	4	0101	3	101	3	110
O	0.07245796	4	0100	4	1001	3	101	3	101	4	1010	4	0010	4	0010
R	0.06872164	4	0101	4	0000	4	0110	4	0110	4	0010	4	0011	4	0011
N	0.06498532	4	0110	4	1111	4	1001	4	1001	4	1101	4	0110	4	0100
H	0.05831331	4	1000	5	01110	5	00100	5	00100	4	0100	4	0111	4	0111
I	0.05644515	4	1001	5	10001	5	11011	5	11011	4	1011	4	1110	4	1001
S	0.05537763	4	1010	5	00100	5	01110	5	01110	4	0110	4	1111	4	1111
D	0.04376834	5	00010	5	11011	5	10001	5	10001	5	11001	5	01001	5	01010
L	0.04123298	5	00011	6	011110	6	001100	6	001100	5	10011	5	01010	5	01011
U	0.02762209	5	10110	6	100001	6	110011	6	110011	5	01110	5	01011	5	01100
P	0.02575393	5	10111	6	001100	6	011110	6	011110	5	10001	5	11001	5	10001
F	0.02455297	5	11100	6	110011	6	100001	6	100001	6	001100	5	11010	5	11100
M	0.02361889	5	11110	7	0111110	7	0010100	7	0010100	6	011110	5	11011	6	011010
C	0.02081665	5	11111	7	1000001	7	1101011	7	1101011	6	100001	6	010001	6	011011
W	0.01868161	6	011100	7	0010100	7	0011100	7	0011100	7	1001001	6	110001	6	100001
G	0.01521216	6	011101	7	1101011	7	1100011	7	1100011	7	0011100	7	0100001	6	111010
Y	0.01521216	6	011110	7	0011100	7	0111110	7	0111110	7	1100011	7	1100001	6	111011
B	0.01267680	6	011111	7	1100011	7	1000001	7	1000001	7	0111110	8	01000001	7	1000001
V	0.01160928	6	111011	7	0001000	8	00111100	8	00111100	7	1000001	8	11000001	8	10000001
K	0.00867360	7	1110100	7	1110111	8	11000011	8	11000011	8	00111100	9	010000001	9	100000001
X	0.00146784	8	11101011	8	01111110	8	01111110	8	01111110	8	11000011	9	110000001	1	1000000001
J	0.00080064	9	111010101	9	011111110	8	10000001	8	10000001	9	100101001	1	0100000001	1	10000000001
Q	0.00080064	1	111010100	1	011111111	9	011111110	9	011111110	1	001110100	1	1100000001	1	100000000001
Z	0.00053376	1	111010100	1	100000000	9	100000001	9	100000001	1	100101110	1	0100000000	1	100000000000
Average length		4.15572392		4.60728507		4.46463681		**4.46463681**		4.30677804		4.18734808		**4.172804**	