Signal Processing: Image Communication, 20(2), 2005, pp. 187-192.

On decoding MPEG–4 reversible variable length codes

Kuo–Liang Chung[a,*], *Senior Member, IEEE* and Hsiu–Niang Chen[b,†]

[a]*Department of Computer Science and Information Engineering*

[b]*Department of Information Management*

National Taiwan University of Science and Technology

No. 43, Section 4, Keelung Road, Taipei, Taiwan 10672, R.O.C.

Abstract

Recently, Webb presented some efficient indexing methods for decoding reversible variable length codes (RVLCs) which are used to encode MPEG–4 DCT coefficients. Given an RVLC, this paper presents two new indexing methods for decoding the RVLCs of MPEG–4 DCT coefficients. Compromising the time requirement, our proposed decoding methods need the least row memory requirement when compared to Webb's methods.

Keywords: DCT coefficients; Error resilience; MPEG–4; Reversible variable length codes

1. Introduction

Variable length codes (VLCs) have been used in the entropy coding phase in some image coding standards to improve the compression rate. Using VLCs in a noisy environment, even a single bit error, the received codes may become useless. Since reversible VLCs (RVLCs) can be decoded in forward and backward directions, they have been suggested in H.263+, H.263++ [2], and MPEG–4 [1] to enhance their error resilience capabilities [4, 5]. Webb [3] presented four efficient indexing methods for decoding RVLCs which are used to encode MPEG–4 DCT coefficients. MPEG–4 RVLC B–23 table is used to save

*Corresponding author. *E–mail address:* klchung@cs.ntust.edu.tw (K. L. Chung), supported by the National Science Council of R.O.C. under contract NSC92-2213-E011-079.

†The author is also with Department of Information Management, Vanung University of Science and Technology, No. 1, Vannung Road, Shuiwei, Chungli, Taiwan 320, R.O.C.

the last–run–level values of the MPEG–4 DCT coefficients.

Without needing any extra row memory requirement, but compromising the time requirement, this paper first presents a new indexing method for decoding RVLCs in the MPEG–4 RVLC B–23 table. Next we propose a faster decoding method, but need an extra row memory. Some experiments are carried out to compare the performance among our two methods and Webb's four methods.

2. Past work by Webb

The RVLCs are used to encode DCT coefficients in the MPEG–4 RVLC B–23 table (see Table 1). In the last column of Table 1, there are 169 RVLCs and each RVLC consists of a RVLC prefix concatenated with a 2–bit suffix. After examining these 169 RVLCs, the RVLC prefix either starts and ends in a 1 with only 0s (if any) in the middle, or starts and ends in a 0 with exactly one 0 and 1s (if any) in the middle. Due to the special structure of these codewords, i.e. RVLCs, using the conventional VLC decoding method to decode RVLCs is not an efficient way. Recently, Webb [3] presented four efficient indexing methods to decode RVLCs. In what follows, we briefly introduce Webb's four efficient indexing methods.

If the received codeword begins with 1, any one of Webb's four methods applies the following formula to obtain the table index of the received codeword, say $index_{w0}$:

$$index_{w0} = 2 \times n_0 + b \tag{1}$$

where n_0 denotes the number of 0's in the RVLC part and b denotes the value of the first bit in the 2–bit suffix part. Since the value of n_0 ranges from 0 to 11 and b is either 0 or 1, the value of $index_{w0}$ ranges from 0 to 23 with no gaps. For example, if the received codeword is '110s' where s denotes the value of the sign bit, by Eq. (1), we have $index_{w0} = 2 \times 0 + 0 = 0$. In addition, if the received codeword is '10000000000011s', then we have $index_{w0} = 2 \times 11 + 1 = 23$.

If the received codeword begins with 0, in what follows, Webb presents four efficient indexing methods, namely Type_1, Type_2, Type_3, and Type_4. In fact, throughout this

paper, Webb's Type_i method consists of the decoding method for the received codeword beginning with 1 (mentioned in the last paragraph) and beginning with 0 (mentioned in the remaining paragraphs of this section).

A. Type_1:

Webb's first method applies the following formula to derive the table index $index_{w1}$:

$$index_{w1} = 32 \times n_1 + 2 \times n_2 + b \tag{2}$$

where n_1 denotes the number of 1's in the RVLC part ; n_2 denotes the number of 1's before the 2nd 0 in the RVLC part, and b has been defined in Eq. (1). When the value of n_1 ranges from 0 to 10, n_2 ranges from 0 to n_1. Specifically, when n_1 is 11, n_2 ranges from 0 to 6. However, if n_1 equals to 0, b has only a '1'. Consequently, the possible values of $index_{w1}$ are $1, 32, 33, 34, 35, ..., 365$. For example, if the received codeword is '0001s', we have $index_{w1} = 32 \times 0 + 2 \times 0 + 1 = 1$. If the received codeword is '011111101111101s', we have the maximal index value $index_{w1} = 32 \times 11 + 2 \times 6 + 1 = 365$. Although $index_{w1}$ can be formed by concatenating (shifting and or–ing) n_1 (4 bits) with n_2 (4 bits) and b (1 bit), the value of $index_{w1}$ ranges from 0 to 365. However, only 145 (=169-24) row entries are used. Therefore, Webb's Type_1 method needs 390 (=366+24) row entries, but 221 (=390-169) row entries are wasted. In Table 1, the sixth column denotes the mapped indices by using Eq. (1) and Eq. (2) based on Webb's Type_1 method.

B. Type_2:

In order to reduce the amount of extra row entries in the table, Webb's second method applies Eq. (3) to get the table index $index_{w2}$:

$$index_{w2} = 32 \times n_1 + 2 \times n_2 + b$$
$$\text{if } (index_{w2} \geq 192) \;\; index_{w2} = 383 - index_{w2}. \tag{3}$$

Using Eq. (3) to reduce gaps in the table entries, the value of $index_{w2}$ ranges from 0 to 191, but 47 (=192-145) row entries are wasted. For example, if the received codeword

is '0011111100s', we first have $index_{w2} = 32 \times 6 + 2 \times 0 + 0 = 192$. Then, we have the folded index value, $index_{w2} = 383 - 192 = 191$, which is the maximal index value in this method. As a result, Webb's Type_2 method needs 216 (=192+24) row entries, but 47 (=216-169) row entries are wasted. The mapped indices are shown in the seventh column of Table 1.

C. Type_3:

Webb's third method applies the following formula to obtain the table index $index_{w3}$:

$$index_{w3} = 22 \times n_1 + 2 \times n_2 + b. \tag{4}$$

It is easy to check that $index_{w3}$ ranges from 0 to 255, but 111 row entries are wasted. For example, if the received codeword is '011111101111101s', we have $index_{w3} = 22 \times 11 + 2 \times 6 + 1 = 255$. Consequently, Webb's Type_3 method needs 280 (=256+24) row entries, but 111 (=280-169) row entries are wasted. In Table 1, the eightieth column denotes the mapped indices.

D. Type_4:

In Webb's last method, Webb applies the following formula to get the table index $index_{w4}$:

$$index_{w4} = 26 \times n_1 + 2 \times n_2 + b$$
$$\text{if } (index_{w4} \geq 156) \;\; index_{w4} = 311 - index_{w4}. \tag{5}$$

Using the above reordering method, the gaps in the table entries can be reduced significantly. The value of $index_{w4}$ ranges from 0 to 155 and 11 entries are wasted. For example, if the received codeword is '0011111100s', we first have $index_{w4} = 26 \times 6 + 2 \times 0 + 0 = 156$. Then, we have $index_{w2} = 311 - 156 = 155$. Therefore, Webb's Type_4 method needs 180 (=156+24) row entries, but 11 (=180-169) row entries are wasted. In Table 1, the ninth column denotes the mapped indices by using Eq. (1) and Eq. (5).

3. The proposed decoding method

In any one of Webb's methods, there are at least 11 wasted row entries. In this section, given a codeword, we first present a new formula to obtain the corresponding table index without needing any extra row entry. Then, we present a faster method, but need an extra row entry. For convenience, our proposed first method is called Ours(1) and the proposed second method is called Ours(2).

A. Ours(1):

If the received codeword begins with 1, we apply the following formula to obtain the table index, say $index_{c1}$:

$$\begin{aligned} &\text{if } (n_0 = 0) \;\; index_{c1} = b \\ &\text{else } \; index_{c1} = n_0 \times (n_0 + 3) + b - 1 \end{aligned} \tag{6}$$

where n_0 and b have been defined in Section 2 and have the same ranges as in Eq. (1). The possible values of $index_{c1}$ are $0, 1, 3, 4, ..., 153$, and 154. These mapped indices are consistent with the original table indices. For example, if the received codeword is '110s', we have $index_{c1} = b = 0$. If the received codeword is '10000000000011s', we have $index_{c1} = 11 \times (11 + 3) + 1 - 1 = 154$.

If the received codeword begins with 0, we use the following formula to derive the table index $index_{c1}$:

$$index_{c1} = (n_1 + 1) \times (n_1 + 2) + 2 \times n_2 + b - 1 \tag{7}$$

where n_1, n_2 and b have the same definitions and ranges as in Eq. (2). Thus, the $index_{c1}$ in Eq. (7) has the possible values $2, 5, 6, 7, 8, ..., 167$, and 168. These mapped indices are consistent with the original table indices. For example, if the received codeword is '0001s', we have $index_{c1} = (0 + 1) \times (0 + 2) + 2 \times 0 + 1 - 1 = 2$. If the received codeword is '011111101111101s', we have $index_{c1} = (11+1) \times (11+2) + 2 \times 6 + 1 - 1 = 168$. In Table 1, the tenth column denotes the mapped indices by using Eq. (6) and Eq. (7). Therefore,

using the above mapped indices to decode the MPEG–4 RVLC B–23 table, no gap is in the table entries. On the other hand, the proposed Ours(1) method only needs 169 row entries without any extra row entry required.

B. Ours(2):

In order to reduce the execution time requirement, we further modify the formulas in Eq. (6) and Eq. (7) to obtain the new table index $index_{c2}$. If the received codeword begins with 1, we apply the following formula to obtain the table index $index_{c2}$:

$$index_{c2} = n_0 \times (n_0 + 3) + b. \tag{8}$$

Since the value of n_0 ranges from 0 to 11 and b is either 0 or 1, the possible values of $index_{c2}$ are $0, 1, 4, 5, ..., 154$, and 155.

If the received codeword begins with 0, we apply the following formula to obtain the table index $index_{c2}$:

$$index_{c2} = (n_1 + 1) \times (n_1 + 2) + 2 \times n_2 + b. \tag{9}$$

It is easy to verify that the possible values of $index_{c2}$ are $3, 6, 7, 8, 9...168$, and 169. Using the proposed modified method, only one extra row entry is needed. The new table index $index_{c2}$ can be viewed as shifting one position after the index 1. As a result, our proposed Ours(2) method needs 170 row entries with only one wasted row entry. In Table 1, the eleventh column illustrates the mapped indices of our proposed second method.

4. Experimental results

Some experiments are carried out to evaluate the performance among Webb's four methods and our proposed two methods. The used machine is Pentium III PC with 450 MHz and the used language is C language. To assure a fair comparison between Webb's indexing methods and our indexing methods, all of the unknown variables and the concerned branches are generated by the random number generator.

Table 2 shows the concerned equations and the operation types required in Webb's four indexing methods and our two indexing methods. Table 3 shows the memory requirement, i.e. the row entries in the table, and the time requirement for the six methods. The time unit is 10^{-9} second. In our indexing methods, the Ours(1) method has no redundancy in the table entries and the Ours(2) method has only one extra row entry in the table but it is faster than the proposed Ours(1) method. However, any one of Webb's methods has at least 11 extra row entries in the table. Specifically, our two indexing methods need less execution time when compared to Webb's Type_4 method.

5. Conclusion

RVLCs have been suggested in the emerging video coding standards to enhance their error resilience capabilities in a noisy environment. Given an RVLC, this paper has presented two new indexing methods for decoding the RVLCs of MPEG–4 DCT coefficients. Under reasonable execution time requirement, in our two methods, the first method has no redundancy in the table entries and the second method has only one extra row entry in the table. However, there are at least 11 extra table entries in any one of Webb's methods.

Efficient decoding in terms of both memory and time requirements is critical for the noisy environment such as low-power wireless devices. According to the memory and time comparison in Table 3, the users are suggested selecting Webb's Type_1 method or Type_2 method due to the computational advantage if the decoder doesn't provide fast multiplication capability. Otherwise, if the decoder provides fast multiplication capability, the users are suggested selecting any one of our proposed two methods, Ours(1) and Ours(2) due to the memory–saving advantage and the modest time requirement. In other words, the users can determine the best way to decode the RVLCs according to their requirements and the used decoder environments.

References

[1] ISO/IEC Final Draft International Standard 14496–2, Coding of audio–visual objects: Visual, Oct. 1998.

[2] ITU–T Recommendation H.263, Video coding for low bit rate communication, 1997.

[3] Jennifer L. H. Webb, Efficient table access for reversible variable-length decoding, IEEE Trans. Circuits and Systems for Video Technology 11 (2001) 981–985.

[4] J. Wen, J. D. Villasenor, A class of reversible variable length codes for robust image and video coding, Proc. IEEE Int. Conf. Image Processing 2 (1997) 65–68.

[5] J. Wen, J. D. Villasenor, Reversible variable length codes for efficient and robust image and video coding, Data Compression Conference (1998) 471–480.

Table 1
Indices of MPEG–4 RVLC B–23 table for decoding DCT coefficients

B–23 *index*	n_0	n_1	n_2	b	Type_1	Type_2	Type_3	Type_4	Ours(1)	Ours(2)	codeword
0	0			0	(0)	(0)	(0)	(0)	0	0	11 0s
1	0			1	(1)	(1)	(1)	(1)	1	1	11 1s
2		0	0	1	1	1	1	1	2	3	000 1s
3	1			0	(2)	(2)	(2)	(2)	3	4	101 0s
4	1			1	(3)	(3)	(3)	(3)	4	5	101 1s
5		1	0	0	32	32	22	26	5	6	0010 0s
6		1	0	1	33	33	23	27	6	7	0010 1s
7		1	1	0	34	34	24	28	7	8	0100 0s
8		1	1	1	35	35	25	29	8	9	0100 1s
9	2			0	(4)	(4)	(4)	(4)	9	10	1001 0s
10	2			1	(5)	(5)	(5)	(5)	10	11	1001 1s
11		2	0	0	64	64	44	52	11	12	00110 0s
12		2	0	1	65	65	45	53	12	13	00110 1s
13		2	1	0	66	66	46	54	13	14	01010 0s
14		2	1	1	67	67	47	55	14	15	01010 1s
15		2	2	0	68	68	48	56	15	16	01100 0s
16		2	2	1	69	69	49	57	16	17	01100 1s
17	3			0	(6)	(6)	(6)	(6)	17	18	10001 0s
18	3			1	(7)	(7)	(7)	(7)	18	19	10001 1s
19		3		0	96	96	66	78	19	20	001110 0s
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
55		6	0	0	192	191	132	155	55	56	001111110 0s
56		6	0	1	193	190	133	154	56	57	001111110 1s
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
152		10	10	1	341	42	241	30	152	153	0111111111100 1s
153	11			0	(22)	(22)	(22)	(22)	153	154	1000000000001 0s
154	11			1	(23)	(23)	(23)	(23)	154	155	1000000000001 1s
155		11	0	0	352	31	242	25	155	156	00111111111110 0s
156		11	0	1	353	30	243	24	156	157	00111111111110 1s
157		11	1	0	354	29	244	23	157	158	01011111111110 0s
158		11	1	1	355	28	245	22	158	159	01011111111110 1s
159		11	2	0	356	27	246	21	159	160	01101111111110 0s
160		11	2	1	357	26	247	20	160	161	01101111111110 1s
161		11	3	0	358	25	248	19	161	162	01110111111110 0s
162		11	3	1	359	24	249	18	162	163	01110111111110 1s
163		11	4	0	360	23	250	17	163	164	01111011111110 0s
164		11	4	1	361	22	251	16	164	165	01111011111110 1s
165		11	5	0	362	21	252	15	165	166	01111101111110 0s
166		11	5	1	363	20	253	14	166	167	01111101111110 1s
167		11	6	0	364	19	254	13	167	168	01111110111110 0s
168		11	6	1	365	18	255	12	168	169	01111110111110 1s

Table 2
Concerned equations and operation types required in the six methods

method	concerned equations	operation types
Type_1 [3]	1–start: $index_{w0} = 2 \times n_0 + b$ 0–start: $index_{w1} = 32 \times n_1 + 2 \times n_2 + b$	2 assignments, 3 shifts, 3 or's
Type_2 [3]	1–start: $index_{w0} = 2 \times n_0 + b$ 0–start: $index_{w2} = 32 \times n_1 + 2 \times n_2 + b$ if $(index_{w2} \geq 192)$ $index_{w2} = 383 - index_{w2}$	2 or 3 assignments, 3 shifts, 3 or's, 1 if, 0 or 1 subtraction
Type_3 [3]	1–start :$index_{w0} = 2 \times n_0 + b$ 0–start: $index_{w3} = 22 \times n_1 + 2 \times n_2 + b$	2 assignments, 3 additions, 3 multiplications
Type_4 [3]	1–start: $index_{w0} = 2 \times n_0 + b$ 0–start: $index_{w4} = 26 \times n_1 + 2 \times n_2 + b$ if $(index_{w4} \geq 156)$ $index_{w4} = 311 - index_{w4}$	2 or 3 assignments, 3 additions, 3 multiplications, 1 if, 0 or 1 subtraction
Ours(1)	1–start: if $(n_0 = 0)$ $index_{c1} = b$ else $index_{c1} = n_0 \times (n_0 + 3) + b - 1$ 0–start: $index_{c1} = (n_1 + 1) \times (n_1 + 2) + 2 \times n_2 + b - 1$	2 assignments, 4 or 6 additions, 2 or 3 multiplications, 1 if else, 1 or 2 subtractions
Ours(2)	1–start: $index_{c2} = n_0 \times (n_0 + 3) + b$ 0–start: $index_{c2} = (n_1 + 1) \times (n_1 + 2) + 2 \times n_2 + b$	2 assignments, 6 additions, 3 multiplications

Table 3
Memory and time comparison

method	memory		time
	table size	No. of wasted entries	
Type_1 [3]	390	221	5.16
Type_2 [3]	216	47	14.61
Type_3 [3]	280	111	10.27
Type_4 [3]	180	11	18.29
Ours(1)	169	0	16.80
Ours(2)	170	1	14.29