論文2001-38SP-3-8

가역가변길이 부호를 위한 테이블 압축방법

(A Table Compression Method for Reversible Variable Length Code)

任 善 雄 *, 裵 皇 植 *, 鄭 正 和 *

(Sun-woong Im, Hwang-sik Bae, and Jong-wha Chong)

요 약

본 논문에서는 가역가변길이 부호를 테이블 메모리에 효율적으로 저장하는 방법을 제안한다. 여러개의 부호들을 적은 개수의 값들로 테이블을 구성하는 새로운 알고리듬으로, 가역가변길이 부호의 부호내의 비트 천이 개수와 부호 구성 트리에서의 레벨을 이용하는 TNWT(Transition Number and Weight of Tree)방법을 제안한다. 압축에 앞서 가역가변길이 부호들의 가중치와 천이개수를 구하고, 신장된 값들이 서로 구분이 안되는 경우를 방지하기 위해 테이블의 값들을 재배열한다. 재배열이 끝난 배열의 값들을 세 개씩 묶어 압축된 테이블을 얻는다. 압축된 테이블은 부호의 천이개수와 가중치를 이용하여 복호해 낼 수 있다. 이러한 방법을 통하여 기존의 방법보다 약20% 적은 크기로 테이블 메모리를 구성하고, 압축된 테이블로 복호가 가능함을 확인하였다.

Abstract

A table compression method for reversible variable length code is proposed in this paper. TNWT(Transition Number and Weight of Tree) method, which uses the transition number of bits within a symbol and the level of a code tree, is proposed. Compression of table values is performed after arrangment of values that is not distinghishable by transition number and weights. In decoding, the transition number and weight of code are used. In this method, the table for RVLC decoding can be implemented with a smaller memory.

Ⅰ. 서 론

최근 첨단기술의 발전으로 인한 이동 통신의 급속한 성장은 영상, 음성등의 멀티미디어 데이터의 신뢰성 있는 전송과 늘어난 정보량을 보다 효율적으로 처리할 수 있는 기술을 요구하게 되었다.

비디오 압축기술은 대역폭이 제한된 채널에서 영상정보를 보내게 해주며, 이러한 기술중의 하나로 가변길이 부호화(Variable Length Coding)가 채택되는데, 가변길이 부호화의 경우 대부분 허프만 부호화(Huffman Coding)가 쓰인다. 그리고 최근 MPEG-4등에서 채택하고 있는 가역가변길이 부호(Reversible Variable Length Code)는 기존의 가변길이 부호에 가역성(reversibility)를 추가한 것이다. 부호의 가역성은 기존의 가변길이 부호에서 오류가 발생했을 때 버려야 했던 데이터들 중 상당 부분을 복원해낼수 있는 장점이 있다.

가역가변길이 부호를 복호할 때 필요한 룩업테이블은 복호시 필요한 모든 코드워드(codeword)에 대한 기술을 갖고 있어야 하므로 많은 양의 메모리를 필요로

* 正會員, 漢陽大學校 電子工學科 CAD 및 通信回路 研究室

(CAD & C.C. Lab., Dept. of Electronic Eng., Hanyang Univ.)

接受日字:2000年11月16日, 수정완료일:2001年1月19日

한다. 따라서, 복호기를 하드웨어로 구현시 비교적 많은 자원(resources)를 차지하게 된다[10]. 본 논문에서는 효율적인 테이블 구성 방법을 통하여, 테이블 메모리의 양을 최적화하는 방법을 제안한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 가역가변길이 부호를 기술하고 3장에서는 TNWT를 이용한 가역 가변길이 부호 테이블 압축방법을 제안하고 성능향상에 대해 기술한다. 그리고, 4장에서는 결론을 맺는다.

Ⅱ. 가역가변길이 부호

허프만 부호화(Huffman coding)로 알려진 가변길이 부호화는 JPEG, MPEG, 기타 이미지 데이터 표준에서 널리 사용된 데이터 압축 방법중의 하나이다. 가변길이 부호화의 기본원리는 이산여현변환(DCT, Discrete Cosine Transform)이나 양자화(quantization)를 수행한 후에 통계적인 잉여항(statistical redundancy)를 제거하는 것이다.

이러한 가변길이 부호들은 복호과정중 오류(error)가 발생했을 때 그림 1에서처럼 동기(synchronization)를 잃게 되고 결과적으로 다음 동기까지의 모든 데이터를 복호할 수 없다는 문제점을 갖고 있다. 이러한 문제점의 대안으로 가역가변길이 부호가 제안되었다.

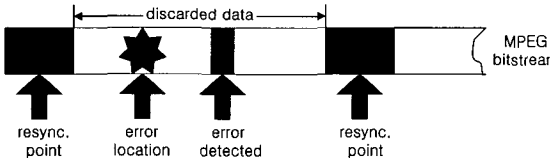


그림 1. 일반적인 가변길이 부호(VLC)의 오류처리
Fig. 1. Error handling in conventional VLC

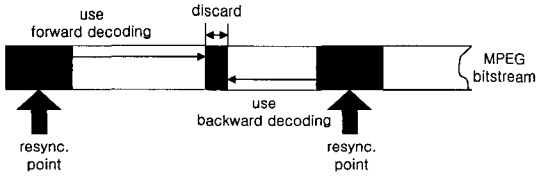


그림 2. 가역가변길이 부호(RVLC)의 오류처리
Fig. 2. Error handling in RVLC.

MPEG-4의 오류 내성 구조중의 하나로 채택된 가역가변길이 부호는 기존의 가변길이 부호에서 불가능했던 역방향 복호(backward decoding)가 가능하다. 그림 2에서 처럼 역방향 복호를 통해서 기존의 가변 길이 부호에서 잃을 수 밖에 없던 데이터의 상당 부분을 복구할 수 있다[3].

가역 가변길이 부호의 생성조건은 접두부호조건과 접미부호조건 두가지가 있다. 접두부호 조건은 각각의 부호가 자신보다 더 긴 부호의 앞부분과 일치하면 안되는 것이고, 이 조건은 기존의 비가역 가변길이 부호에도 적용된다. 접미부호 조건은 각각의 부호가 자신보다 더 긴 부호의 뒷부분과 일치하면 안 되는 것인데, 이 조건은 가역 가변길이 부호(RVLC)의 역방향 복호(backward decoding)시에 올바른 부호로의 해석을 보장해주기 위한 조건이다.

가역가변길이부호는 코드의 대칭성을 기준으로 대칭 가역가변길이 부호(symmetric RVLC)와 비대칭 가역가변길이 부호(asymmetric RVLC)로 나뉜다[1].

대칭 가역가변길이부호는 메모리 사용량과 부호의 간결함의 측면에서 좋은 성능을 나타내고 가변길이 부호에서 앞에서 언급한 가역가변길이부호 생성조건을 만족시켜 만들어낸다. 이 과정은 다음과 같이 그림 3과 그림 4를 통해 설명될 수 있다.

그림 3의 (a)는 5개의 부호로 구성된 간단한 비가역 가변길이 부호를 나타내는 트리의 한 예이며, 이를 가역가변길이 부호로 만들어 갈 것이다. 부호 A는 '00'인데, 대칭이며 생성조건을 만족한다. 부호 B는 '01'이며 대칭성을 만족시키지 못하므로, 트리에서 다른 노드로 바꿔줘야 하는데, 먼저 그림 4에서 동일한 레벨을 검사한다. 그런데, 같은 레벨의 '11'은 이미 C로 할당되어 있으므로, 다음 레벨에서 '000'부호를 검사한다. 이 부호는이미 부호 A로 선택된 '00'때문에 부호의 생성조건에 위배된다. 따라서, 다음 부호인 '010'을 B로 할당한다. 부호 C는 대칭이며 부호의 생성조건을 만족시킨다. 비대칭인 부호 D는 대칭인 부호로 바꿔줘야 하는데, 같은 레벨의 '101'은 이미 부호 E로 할당되어 있고 '111'은 부호 C로 인해 부호의 생성조건에 위배되므로 다음 레벨을 검사한다. 다음 레벨에서 부호 '0000'은 부호 A와 생성조건에 위배되므로, 그 다음 대칭 부호인 '0110'을 부호 D로 할당한다. 그리고 부호 E는 '101'로 대칭이며, 부호의 생성조건에 위배되지 않고, 부호 레이블 정렬(label sorting)을 수행함으로써 최종적인 결과인 그림 3(d)를 얻는다.

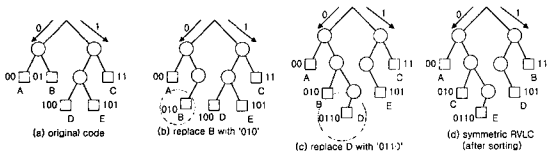


그림 3. 대칭 가역가변길이 부호 생성 예
Fig. 3. Example of Symmetrical RVLC generation.

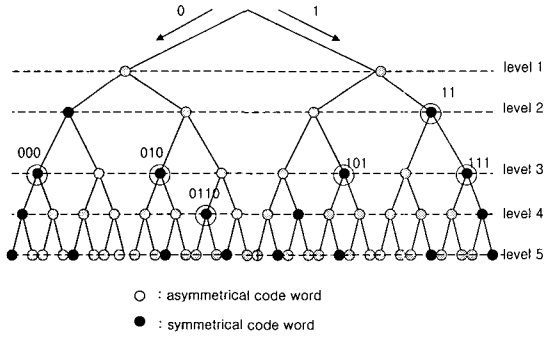


그림 4. 대칭 가역가변길이 부호 이진 트리
Fig. 4. Binary-tree of Symmetrical RVLC.

비대칭 가역가변길이 부호는 대칭성을 고려하지 않고 부호를 생성하기 때문에 부호 생성시 대칭 가역가변길이 부호보다 더 좋은 효율을 나타낸다. 비대칭 가역가변길이 부호 역시 비가역가변길이 부호에서 가역가변길이 부호의 생성조건을 검사하여 만들어낸다.

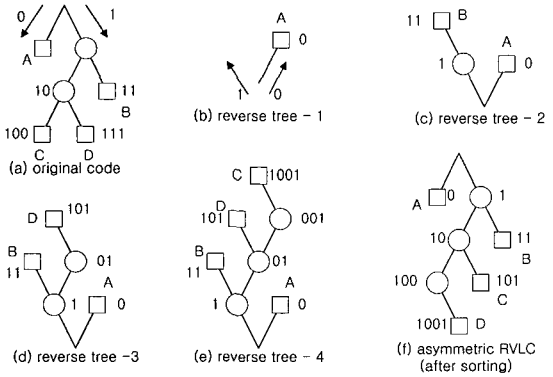


그림 5. 비대칭 가역가변길이 부호 생성 예
Fig. 5. Example of Asymmetrical RVLC generation.

그림 5를 보면서 생성과정을 설명해 보면 다음과 같다. 그림 5(a)는 4개의 부호로 구성된 기존의 비가역가변길이 부호의 한 예이다. 그림 5(b)처럼 역 이진 트리를 만든다. 그 다음 부호를 따라가면서, 각 부호의 접미부호 조건을 검사하여 위배될 경우 새로운 부호를 할당함으로써 비대칭 가역가변길이 부호를 만들게 된다. 그림 5에 있는 부호의 경우 부호 C의 경우가 그러하다. '001'이란 부호를 C로 할당하면, 부호 A가 부호 C의 접두부호와 동일하게 되므로 다른 부호로 할당해주어야 한다. 같은 레벨에 할당할 수 있는 부호가 없을 때는 최소수의 비트(bit)를 추가해줌으로써 새로운 부호를 할당한다. 이 경우 '1001'의 부호를 부호 C로 할당할 수 있다. 부호의 할당이 끝난 다음 대칭 가역가변길이 부호의 경우와 마찬가지로 정렬을 수행함으로써, 비대칭 가역가변길이 부호의 생성을 완성한다.

Ⅲ. TNWT를 이용한 가역가변길이 부호 테이블 압축

일반적인 가변길이부호의 메모리 효율성을 고려한 압축 알고리듬에는 부호의 앞부분의 패턴(Pattern)이 유사한 점을 이용한 연구들[4]과, 허프만-트리의 레벨과 부호값을 이용한 연구들[2,5]이 있다. 이러한 복호 알고리듬은 주로 가변길이의 부호간 유사성과 허프만-트리(Huffman-tree)가 완전-트리(Complete-tree)인 점을 이용하여 진행되었다[2,4,5]. 가역가변길이 부호의 생성과정을 살펴보면, 이를 트리로 구성할 경우에 트리가 희소(Sparse)해지는 경향을 관찰할 수 있다. 그리고, 부호간의 유사성도 찾기 힘들다. 따라서, 가역가변길이 부호의 경우 상기 기술된 방법은 적용하기 힘들며 다른 접근방법이 필요하다.

1. 테이블 구성과 압축

Ⅱ에서와 같이 가역가변길이 부호로 완성된 부호들은 복호를 위해 테이블 메모리에 저장되어야 한다. 본 논문에서는 테이블을 압축하는 방법으로 각 부호들의 가중치(weight)와 부호내의 천이개수(transition number)를 이용하는데, 이를 TNWT(Transition Number and Weight of Tree)이라 칭하기로 한다.

테이블 메모리는 다음과 같이 구성된다. 먼저 각각의 부호들의 가중치와 천이개수를 구하고 각각을 2의 멱승(power)형태로 만든후, 가중치를 기준으로 내림차순(descending)으로 테이블로 사용할 배열에 저장한다. 표 2는 이렇게 구해진 가중치와 천이개수를 배열로 저장한 결과이다. 가중치와 천이개수는 복호시 부호를 구별하는 기준으로 삼게 되는데, 이러한 값들이 동일하여 구분이 안 되는 부호들을 위해서 값들의 재배열이 필

요하다. 표 3이 이러한 재배열을 수행한 결과를 나타내며 각각의 배열값에 대하여 세 개의 값을 하나로 묶어 저장하여 압축된 테이블을 구하게 된다. 표 4는 이렇게 완성된 테이블이다.

2. TNWT(Transition number and Weights of Tree)

TNWT를 이용한 압축방법은 가중치와 천이개수를 사용한다.

가중치는 테이블을 구성하는 가역가변길이 부호 트리의 가장 큰 레벨의 값에서 자신의 레벨을 뺀 값을 2의 멱승형태로 표현한 값이다. 식으로는 다음과 같이 표현된다.

$$w = 2^{(h-l)},$$
$$(h: height\ (highest\ level),\ l: level) \quad (1)$$

예를 들어 어떤 가변길이 복호에 사용된 테이블의 가장 긴 코드의 레벨이 7이라고 한다면 레벨이 2인 코드의 가중치는 $2^{(7-5)} = 4$ 가 된다. 본 논문에서 사용한 가역가변길이 부호 중 가장 긴 부호의 길이는 표 1에서 보면 부호 'Z'의 경우로 '13'이다. 따라서, 각각의 부호들의 가중치는 '13'에서 자신의 부호길이를 뺀 값을 2의 멱승형태로 구하면 된다.

천이개수는 각각의 부호마다 두가지 형태의 값을 구한다.

첫 번째 형태의 천이개수(transition no. type I)는 부호내의 비트 변화수를 나타내는 것으로 다음과 같이 구한다. '01110011'이라는 부호의 경우, '0'에서 '1'로 '1'에서 '0'으로 천이되는 개수는 3이다. 따라서 부호 '01110011'의 천이개수는 '3'이 된다. 식으로는 다음과 같이 표현된다.

$$t_1(x) = \begin{cases} 1, & 0 \to 1 \ or\ 1 \to 0 \\ 0, & 0 \to 0 \ or\ 1 \to 1 \end{cases}, \quad (2)$$

두 번째 형태의 천이개수(transition no. type II)는 먼저 구한 첫 번째 형태의 천이개수중 가장 큰 값을 문턱값(threshold)로 지정하고, 천이될 때 더해지는 값을 달리하여 구한다. '0'에서 '1'로 천이하는 경우는 '2'를 더해주고, '1'에서 '0'으로 천이하는 경우는 '1'을 빼준다. 본 논문에 사용한 테이블에서 첫 번째 형태의 천이개수중 가장 큰 값은 표 1의 transition no. type I을 보면 '6'이다. '6'을 문턱값으로 갖고, 위에 설명한 대로 천이개수 값을 구한다. 부호 '01110011'의 경우 '0'에서 '1'로 천이하는 개수는 '2'이고 '1'에서 '0'으로 천이하는 개수는 '1'이다. 따라서 문턱값 '6'에 '4'를 더하고 '1'을 빼면 '01110011'의 두 번째 형태의 천이개수는 '9'가 된다. 표 1에 각각의 부호와 그리고 해당 가중치와 두가지 형태의 천이개수가 나와있다. 식으로는 다음과 같이 표현된다.

$$t_2(x) = \begin{cases} +2, & 0 \to 1 \\ -1, & 1 \to 0 \end{cases}, \quad (3)$$

3. 테이블 값들의 재배열과 압축

Ⅲ.2에서 제안한 TNWT의 가중치와 천이개수를 복호시 부호 구분의 기준으로 삼게 되는데, 복호과정중 가중치와 천이 개수가 같으면 부호를 구분해 낼 수 없다. 이런 경우에는 서로 구분할 수 있는 특징을 추출하여 천이 개수가 들어갈 자리에 특징값을 대신 넣어 복호시 이용한다.

같은 가중치와 천이 개수를 가진 경우에는 먼저 구분할 수 있는 부호들을 배열의 앞부분에 위치시키고, 다른 부호들과 구분이 모호한 것은 상대적으로 배열의 뒷부분에 위치시켜서 복호시 부호를 잘못 구분하는 일이 없게 한다. 또한, 본 논문에서는 부호내에서 가장 먼저 나오는 '1'의 위치를 이용했다. 같은 가중치와 천이 개수를 가진 부호가 있을 때, 부호에서 가장 먼저 나오는 '1'의 위치를 천이 개수와 함께 더하여 2의 멱승형태의 값으로 한다.

재배열된 값들을 세 개씩 묶어서 하나의 값으로 압축하여 표 4와 같은 결과를 만들 수 있다.

4. 압축부호의 신장(decompression)

테이블 압축시 Ⅲ.1에서와 같이 세 개의 값을 하나로 묶었는데, 다음과 같은 식을 두 번 적용하여 압축된 부호를 다시 세 개로 신장시킬수 있다.

$$2^{\lfloor \log_2 X \rfloor}[2] \quad (4)$$

이는 다음과 같이 표현될 수 있다.

Let X : compressed value, A,B,C : values to be compressed

$X = A + B + C$, $(A \geq B \geq C)$

if $2^{\lfloor \log_2 X \rfloor} \neq X$, $X_1 = 2^{\lfloor \log_2 X \rfloor}$,

$X_2 = X - X_1$ $(X_1 \geq X_2)$

if $2^{\lfloor \log_2 X \rfloor} = X$, $X_1 = X_2 = \frac{X}{2}$

case I) in case of nth bit value of decompose = '0'.

(n : bit position of decompose value)

if $2^{\lfloor \log_2 X_2 \rfloor} = X_2$, $A = X_1$, ,

else $X_3 = 2^{\lfloor \log_2 X_2 \rfloor}$

$A = X_1$, $B = X_3$, $C = X_2 - X_3$

case II) in case of nth bit value of decompose = '1'.

if $2^{\lfloor \log_2 X_1 \rfloor} = X_1$, $A = B = \frac{X_1}{2}$,

$C = X_2$

else $X_3 = 2^{\lfloor \log_2 X_1 \rfloor}$

$A = X_3$, $B = X_1 - X_3$, $C = X_2$

위에서 기술된 decompose 값은 압축부호를 두 번째 분해할 때 X_1, X_2 중 어느 값을 이용해야할지를 가리기 위한 값이다. 표 3에서 가중치 배열은 한 가지 형태(case II)의 경우만 존재하므로 이러한 decompose 값이 필요가 없으며, 압축된 테이블의 decision 배열에만 이 값이 필요하게 된다.

예를 들어 표 4의 가중치 중 '2560(1024+1024+512)'의 경우 식(4)를 적용시키면 '2048'과 '512'로 분해가 되고, 이 경우 case II에만 해당된다. '2048'값을 식(4)에 적용시키면 같은 '2048'의 값이 나오므로 두 개의 '1024'로 분리가 되고, 최종적으로 표 4의 '2048'이라는 값은 표 3 에서처럼 '1024', '1024', '512'로 분리가 된다. 표 3 과 표 4 를 살펴보면 이렇게 구성된 부호들의 압축과 신장을 살펴볼 수 있다.

본 논문에서 제안한 압축된 테이블을 복호하는 알고리듬은 다음과 같이 기술된다.

(RVLC memory effcient decoding algorithm)

1. $i \leftarrow 0$

2. if $2^{\lfloor \log x \rfloor} = W_i$, then $A = B = 2^{\lfloor \log x \rfloor}$

otherwise $A = 2^{\lfloor \log x \rfloor}$ and $B = W_i - A$

$w_1 = 2^{\lfloor \log A \rfloor}$, $w_2 = A - w_1$, $w_3 = B$

if weight of code are one of weights, then go to step 3;

else if i = 8, then code is not symbol of RVLC

else $i \leftarrow i+1$ and go to step 2.

3. decision values are decomposed into 3 values by method used in step 2.

3 values values are d_1, d_2, d_3 ($d_1 > d_2 > d_3$).

trn : transition number of code

if $2^{trn} = d_1$ or ($\log_2 d_1 - trn$)th bit of code = 1,then code is $3i$th symbol of RVLC.

else if $2^{trn} = d_2$ or ($\log_2 d_2 - trn$)th bit of code = 1,then code is ($3i+1$)th symbol of RVLC.

else if $2^{trn} = d_3$ or ($\log_2 d_3 - trn$)th bit of code = 1,then code is ($3i+2$)th symbol of RVLC.

else if i indicates last array of weight, .then go to step 4.

else code is not symbol of RVLC.

4. if $W_i = A + B$ and $2^{trn} = d_1 + d_2 + d_3$

,then code is $3i$th symbol of RVLC.

else if $W_i = A$ *or* B

and { ($2^{trn} = d_1 + d_2$ *or* $d_1 + d_3$

or ($\log_2 (d_1 + d_2) - trn$)th bit of code = 1

or ($\log_2 (d_1 + d_3) - trn$)th bit of code=1)}

,then code is $3i$th symbol of RVLC.

else if { ($2^{trn} = d_2$ *or* d_3

or ($\log_2 d_2 - trn$)th bit of code = 1

or ($\log_2 d_3 - trn$)th bit of code = 1) }

,then code is ($3i+1$)th symbol of RVLC.

else code is not symbol of RVLC.

1단계에서의 i는 복호할 테이블의 index를 가리키고, 0으로 초기화 시켜준다.

2단계에서는 먼저 가중치를 분해하여 복호시 들어온 부호의 가중치와 맞는지 확인한다. 맞을 경우에는, 다음 step으로 진행시켜 decision 배열의 값을 분해하고, 맞지 않을 경우 가중치 배열의 다음 위치로 가서 값을 분해한다. 이것은 가중치가 맞을 때 까지 i값을 증가시키면서 반복 수행한다. 가중치 배열에서 해당 가중치를 찾지 못하는 경우는 올바른 부호가 아니므로 수행을 멈추고 i값을 초기화를 하여 처음으로 되돌아간다.

3단계에서는 가중치가 맞을 때, 천이 개수등으로 구

성된 부호의 값과 일치하는지를 2단계에서와 동일한 방법으로 검사한다. 그리고 배열의 값을 세 개씩 묶어 압축을 수행하는 과정중에 마지막 배열의 값이 하나나 둘로 구성될 경우가 있는데, 그 때에는 4단계로 넘어간다.

4단계에서는 배열의 마지막 값이 하나나 두 개의 값을 압축한 경우에 해당될 때이다. 부호를 확인하는 방법은 3단계와 동일하다.

5. TNWT를 사용한 복호 알고리듬의 평가

표 1의 비대칭 가역가변길이부호가 제안한 알고리듬을 검증한 테이블이다. 표 1은 부호와 두가지 형태의 천이개수 값을 나타낸다.

표 1. 부호와 두가지 형태의 천이개수
Table 1. Codeword and two types of transition no.

alphabet	code	transition no. type I	transition no. type II
E	001	1	8
T	110	1	5
A	0000	0	6
O	0100	2	7
R	0101	3	9
N	1000	1	5
H	1010	3	6
I	10010	3	6
S	01100	2	7
D	00010	2	7
L	00011	1	8
U	10111	2	7
P	11100	1	5
F	11111	0	6
M	111101	2	7
C	101101	4	8
W	011101	3	9
G	111011	2	7
Y	01110011	3	9
B	11101011	4	8
V	111010011	4	8
K	011110011	3	9
X	0111110011	3	9
J	1110101011	6	9
Q	11101010011	6	9
Z	1110101000111	6	9

표 2. 가중치와 천이개수를 나타낸 테이블
Table 2. Table for weights and transition no.

symbol	E	T	A	O	R	N	H	I	S	D	L	U	P	F	M	C	W	G	Y	B	V	K	X	J	Q	Z
weight	1024	1024	512	512	512	512	512	256	256	256	256	256	256	256	128	128	128	128	32	32	16	16	8	8	4	1
no. of transition	1	1	0	2	3	1	3	3	2	2	1	2	1	0	2	4	3	2	3	4	4	3	3	6	6	6

표 3. 압축을 위해 변형된 테이블
Table 3. Rearranged table for compression.

symbol	E	T	N	R	O	A	H	L	P	C	I	F	U	S	D	M	G	W	Y	B	Q	K	V	Z	J	X
2^weight	1024	1024	512	512	512	512	512	256	256	128	256	256	256	256	256	128	128	128	32	32	4	16	16	1	8	8
2^decision	256	32	32	512	128	1	64	256	32	256	8	1	4096	2048	512	1024	512	8	512	256	512	512	256	512	64	8

표 2는 테이블 압축을 위해 원래 부호들의 가중치와 천이 개수를 나타낸다.

표 3은 Ⅲ. 3에서 설명한대로 복호시 부호들을 확실히 구분해내기 위해 원래의 테이블을 재배열하고 특징값을 적용시킨 결과이다.

표 4는 TNWT 방법을 적용해 최종적으로 만들어진 가역가변길이 부호에 사용되는 압축된 테이블이다.

표 4. 압축된 테이블
Table 4. Compressed table.

i	0	1	2	3	4	5	6	7	8
weight	2560	1536	1024	640	768	384	68	33	16
decision	320	641	352	265	6656	1544	1280	1280	72

제안한 알고리듬은 배열의 값을 두 번씩 분해하여 복호해야 하는 단점을 지닌다. 하지만 세 개씩 묶어 저장하였으므로, 계산량에 있어 부호가 균등한 분포로 발생할 경우엔 압축하지 않은 경우와 비슷한 성능을 나타낸다.

그리고 메모리 사용의 측면에서는 기존의결과[2]와 비교하여 약20%향상된 결과를 나타낸다. 압축된 테이블과 원래 심벌을 더한 메모리 사용량을 비교한 결과가 표 5에 나와있다.

표 5. 성능 비교 테이블
Table 5. performance comparison table.

	memory space
Chung's algorithm[8]	2n-3
C.W.L algorithm[2]	$\lceil 3n/2 \rceil + \lceil n/2 \log n \rceil + 1$
Our algorithm	$\lceil 5n/3 \rceil + 3$

Ⅴ. 결 론

본 논문에서는 가역가변길이 부호를 테이블 메모리에 효율적으로 저장하는 방법을 제안한다.

제안한 알고리듬은 TNWT(Transition Number and Weight of Tree)라는 부호내의 비트 천이 개수와 부호 구성 트리에서의 레벨을 이용하는 방법을 통하여 테이블 내의 여러 부호들을 압축하여 적은 개수로 저장한다. 압축에 앞서 가역가변길이 부호들의 가중치와 천이 개수를 구하고, 신장된 값들이 서로 구분이 안되는 경우를 방지하기 위해 테이블의 값들을 재배열한다. 재배열이 끝난 배열의 값들을 세 개씩 묶어 최종적으로 원하는 테이블을 얻는다.

압축된 테이블은 부호의 천이개수와 가중치를 이용하여 복호해 낼 수 있었고, 제안된 방법에 의해 가역가변길이 부호의 테이블을 압축한 결과 부호값과 가중치를 이용한 방법[2]보다 약20% 적은 메모리로 테이블을 구성할 수 있었다.

제안된 방법은 MPEG-4와 기타 이미지 압축 알고리듬에 쓰이는 가역가변길이 부호의 룩업테이블(look-up table)과 2의 멱승으로 구성된 값들을 효율적으로 압축하는 데 쓰일 수 있다.

참 고 문 헌

[1] Y. Takishima, M.Wada, H.Murakami, "Reversible Variable Length Codes," *IEEE trans. Communications*, vol43. NO.2/3/4/, 1995, pp.158~162.

[2] H.C.Chen, Y.L Wang. Y.F Lan., "A memory-efficient and fast Huffman decoding algorithm," *Information Processing Letters*, vol.69, 1999, pp. 119~122.

[3] Raj Talluri, "Error-Resilient Video coding in the ISO MPEG-4 standard," *IEEE Communications Magazine*, June 1998, pp.112~119.

[4] Cheng-Teh Hsieh, Seung P.Kim, "A Concurrent Memory-Efficient VLC Decoder for MPEG Applications," *IEEE trans. Consumer Electronics*, vol.43. NO.3, August, 1996, pp.439~446.

[5] Kuo-Liang Chung, Jung-Gen Wu, "Level-Compressed Huffman Decoding," *IEEE trans. Communications*, vol.47, NO.10, October 1999, pp.1455~1457.

[6] Keshab K.Parhi and Takao Nishitani, *Digital Processing for Multimedia Systems*, Marcel Dekker, p.389~394, 1999.

[7] Luis Ducla-Soares, Fernando Pereia, "Error resilience and concealment performance for MPEG-4 frame-based video coding," *Signal Processing : Image communication* vol.14, 1999, pp.447~472.

[8] Kuo-Liang Chung, "Efficient Huffman decooding," *Information Processing Letters* vol.61, 1997, pp.97~99.

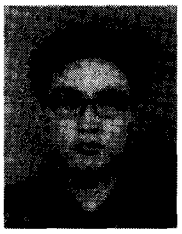
[9] Reza Hashemian, "Memory Efficient and High-Speed Search Huffman Coding," *IEEE trans. Communication*, vol.43, NO.10, October 1995, pp.2576~2581.

[10] Mario Novell and Steve Molloy, "VLSI implementation of a Reversible Variable Length Encoder/Decoder," *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol.4, March 1999, pp.1969~1972.

[11] 이승준, 서기범, 정정화, "A Memory-effcient VLC Decoder Architecture for MPEG-2 Application," 대한전자공학회 추계종합학술대회 논문집, 제22권 제2호, pp 360-363, 1999년 11월

[12] 임선웅, 배황식, 정정화, "효율적인 테이블 메모리를 갖는 가역가변길이 부호," 대한전자공학회 하계종합학술대회논문집, 제23권 제1호, pp.133-136, 2000년 6월

저 자 소 개



任 善 雄(正會員)

1976년 8월 8일생. 1999년 2월 한양대학교 전자·전자통신·전파공학과 군 졸업(공학사). 2001년 2월 한양대학교 대학원 전자공학과 졸업(공학석사). 2001년 2월~현재 닉소텔레콤 연구원. 주관심분야 : 영상 압축

裵 皇 植(正會員)

1972년 7월 15일생. 1995년 2월 한양대학교 전자공학과 졸업(공학사). 1997년 2월 한양대학교 대학원 전자공학과 졸업(공학석사). 1997년 3월~현재 한양대학교 대학원 전자공학과 박사과정. 주관심분야 : 영상 압축 전송

鄭 正 和(正會員) 第 35卷 C編 第 10號 參照

현재 한양대학교 전자전기공학부 교수