블럭정렬과 VF형 산술부호에 의한 오류제어 기능을 갖는 데이터 압축

이 진 호† 조 숙 희†† 박 지 환††† 강 병 욱††††

요 약

본 논문에서는 블럭정렬과 선두 이동법에 의해 처리된 계열을 VF(Variable to Fixed)형 산술부호로 압축하는 방법을 제안한다. 길이 N으로 분해된 부분열을 1기호씩 순회시킨 후 사전식 순서로 정렬한다. 순회정렬된 부분열은 국소적으로 유사기호가 밀집되기 때문에 이 성질을 활용하기 위하여 선두 이동법을 적용한다. 이와 같이 전처리된 계열에 대해 오류전파를 1 부호어 이내로 제한할 수 있는 VF형 산술부호로 엔트로피 부호화 한다. VF형 산술부호의 효율은 고정 크기의 부호어 집합을 어떻게 분할하는가가 관건이다. 제안하는 VFAC(VF Arithmetic Code)는 새로 설정되는 정보원 기호에 대하여 완전분할을 이루게 하고, 반복적인 그레이 변환을 이용하여 발생기호의 확률을 효과적으로 나타낸다. 제안 방식의 성능을 컴퓨터 시뮬레이션을 통하여 엔트로피, 압축율 및 처리속도의 측면에서 기존의 방식과 비교 분석한다.

Data Compression Capable of Error Control Using Block-sorting and VF Arithmetic Code

Jin Ho Lee†, Suk Hee Cho††, Ji Hwan Park††† and Byong Uk Kang††††

ABSTRACT

In this paper, we propose the high efficiency data compression capable of error control using block-sorting, move to front(MTF) and arithmetic code with variable length in to fixed out. First, the substring which is parsed into length N is shifted one by one symbol. The cyclic shifted rows are sorted in lexicographical order. Second, the MTF technique is applied to get the reference of locality in the sorted substring. Then the preprocessed sequence is coded using VF(variable to fixed) arithmetic code which can be limited the error propagation in one codeword. The key point is how to split the fixed length codeword in proportion to symbol probabilities in VF arithmetic code. We develop the new VF arithmetic coding that split completely the codeword set for arbitrary source alphabet. In addition to, an extended representation of symbol probability is designed by using recursive Gray conversion. The performance of proposed method is compared with other well-known source coding methods with respect to entropy, compression ratio and coding times.

1. 서 론

정보통신의 발달에 따라 컴퓨터에서 처리하는 데이터는 대량화 및 다양화되고 있다. 이와 같은 데이터를 효율 좋게 저장하고 전송하기 위한 방법인 데이터 압축의 필요성이 증가됨에 따라 많은 부호들이 연구되고 있다[1]. 이 가운데 VF(Variable to Fixed)형 산술부호는 FV(Fixed to Variable)형인 Huffman부호와는 달리 가변길이 입력에 대해 고정길이의 부호어(codeword)를 출력하는 부호로써, 그 실현이 간단하고 효율이 우수하여 폭넓은 응용이 기대된다.

입력 정보원 기호의 집합 $A=\{a_1, a_2, \cdots, a_m\}$에 대하여 독립적이고, 동일한 확률분포를 갖는 입력계열 $X=x_1\ x_2\ x_3\ \cdots$ 라고 할 때, 각 기호의 발생확률 $p_j=\Pr(x_i=a_j)$, $1\le j\le m$에 대해

† 정 회 원: 경북산업대학교 전자계산학과 교수
†† 준 회 원: 부산수산대학교 전자계산학과 석사과정
††† 정 회 원: 부산수산대학교 전자계산학과 조교수
†††† 정 회 원: 영남대학교 전산공학과 교수
논문접수: 1995년 7월 20일, 심사완료: 1995년 9월 20일

정보원의 엔트로피 $H(X)$는

$$H(X)=-\sum_{i=1}^{m} p_i \log p_i$$

가 된다. FV형 부호는 각 입력기호에 대해 평균적으로 최소한 $H(X)$의 출력비트가 필요하지만, VF형 부호의 경우는 고정길이 출력비트에 대해 평균적으로 최대한 $1/H(X)$의 입력기호를 표현 가능하게 된다. 또한 VF형 부호는 FV형 부호와는 달리 전송오류로 인한 동기(synchronization)를 잃지 않기 때문에 치명적 오류가 발생하지 않으며, 전송오류의 영향이 그 블럭에 한정된다는 장점을 가진다. 엔트로피 부호의 한 방식인 산술부호는 입력계열 전체에 대해 하나의 부호어를 출력하는 스트림 부호(stream code)이다. 그러므로 부호화된 계열의 일부분만을 복호하고자 할 경우에도 처음부터 차례로 복호 알고리즘을 수행하여야 한다. 또한 1비트의 오류가 발생하면 마지막까지 치명적인 영향을 미치게 된다.

최근 Teuholar와 Raita에 의해 제안된 AB-coding[2]은 VF형 산술부호로서 알고리즘이 간단하면서 일반 산술부호에 근접하는 압축효율을 가진다. 그러나 AB-coding의 일반화 방식에 의한 부호화는 블럭부호임에도 불구하고 각 블럭이 완전히 독립적이지 못하기 때문에 블럭부호의 장점인 오류제어 기능을 갖지 못하게 된다. 만약 AB-coding의 기본방식에 따라 부호화하면 각 블럭이 독립적으로 존재하게 되지만, 압축효율은 상당히 떨어지게 된다. 한편, 각 블럭이 완전히 독립적으로 부호화되면서 압축효율이 우수한 블럭산술부호(BAC : Block Arithmetic Coding)가 Chales와 Boncelet에 의해 제안되었다[3]. BAC는 $m=2$인 이진계열에 대해서는 우수한 압축효율을 가지지만 m이 커질수록 압축율과 처리속도가 상당히 떨어지게 된다. 따라서, 큰 m에 대해서도 빠른 처리속도를 유지하면서 우수한 압축율을 가지는 방식이 요구된다.

본 논문에서는 오류제어 기능을 가지면서 우수한 압축율을 달성하는 데이터 압축법을 제안한다. 제안 방식은 기호들간의 상관을 최대로 활용하기 위하여 입력계열에 순회정렬[4]과 선두 이동법[5]을 적용한다. 다음으로 오류 전파를 억제할 수 있도록 2가지의 정보원 모델을 혼합하는 VF형 산술부호를 고안한다. 이때, 산술부호화를 위한 각 기호의 발생확률 표현에 필요한 데이터량을 최소화하기 위하여 그레이 변환과 새로 설계된 부호화 형식을 사용한다. 2장에서는 블럭정렬의 개념과 구체적인 방법을 알아보고, 생성된 계열이 1차 마르코프 정보원에 유효함을 엔트로피 감소의 측면에서 밝힌다. 3장에서는 선두 이동법에 의한 부호화 및 복호법을 나타내며, 블럭정렬에 의한 부분열이 선두 이동법과 어떻게 정합되는지를 밝힌다. 4장은 1차 마르코프 정보원에 적용시키기 위한 VFAC를 제안하며, 그레이 변환을 이용한 새로운 발생확률 표현법을 제시한다. 컴퓨터 시뮬레이션을 통한 제안 방식의 성능을 기존의 방식과 비교 분석하여 5장에 나타낸다.

2. 블럭정렬(Block-sorting)

2.1 부호화

정보원 계열 X를 길이 N의 부분계열 $X_N=x_1 x_2 \cdots x_N$ $x_i \in A$로 분해하여 블럭단위로 처리한다. 이 부분계열 X_N을 1기호씩 순회(cyclic shift)시킨 N개의 서로 다른 순회계열을 사전식 순서로 정렬하여 $N\times N$의 2차원 행렬 M을 작성한다. 순회정렬 계열 $S_k(0\le k<N)$의 마지막 기호 s_k^N을 차례로 연접시키면 길이 N의 새로운 부분계열 L_N이 생성된다. 이때, 모든 S_k는 원래의 부분계열 X_N을 순회시킨 것이므로 L_N을 구성하는 기호는 동일하며 그 위치만이 바뀌게 된다. N개의 순회계열중에는 원래의 부분계열 X_N도 포함되어 있으며, M상의 그 행의 위치 k는 L_N과 더불어 복호에 필요한 송신정보가 된다. 또한, $L_N=g_1 g_2 \cdots g_N$ $g_i \in A$의 i번째 기호는 i번째 순회계열의 마지막 기호 s_i^N에 해당한다.

간단한 예로 4개의 기호로 이루어지는 정보원 알파벳 $A=\{a, b, c, d\}$로부터 발생하는 길이 $N=6$의 부분계열 $X_6=$"$acabdb$"가 블럭정렬되는 과정을 아래에 나타낸다.

(1) $X_6=$"$acabdb$"를 1기호씩 오른쪽으로 순회시켜 얻어진 6개의 순회계열을 사전식으로 정렬하여 6×6의 2차원 행렬 M을 작성한다.

$$M=\begin{bmatrix} S_0 : a\ b\ d\ b\ a\ c \\ S_1 : a\ c\ a\ b\ d\ b \\ S_2 : b\ a\ c\ a\ b\ d \\ S_3 : b\ d\ b\ a\ c\ a \\ S_4 : c\ a\ b\ d\ b\ a \\ S_5 : d\ b\ a\ c\ a\ b \end{bmatrix}$$

(2) 각 순회정렬 계열 S_k의 마지막 기호 S_k^N으로 이루어지는 길이 6의 새로운 부분계열 L_6을 구한다.

$L_6=$"$cbdaab$"

(3) M에서 원래의 부분계열 $X_6=$"$acabdb$"의 위치 k를 구한다.

$k=1$

최종적으로 송신해야 할 정보는 (L_N, k)이나, L_N의 길이는 X_N과 같기 때문에 이 상태로는 압축이 되지 않으며, 오히려 위치정보인 k를 표현하기 위하여 $\lceil \log_2 N \rceil$ 비트만큼 길어지게 된다. 여기서, $\lceil x \rceil$는 x보다 큰 최소정수를 나타낸다. 따라서, 블럭정렬에 의한 L_N을 압축하기 위하여 선두 이동법과 엔트로피 부호화가 뒤따라야 한다.

2.2 복호화

다음으로 수신된 (L_N, k)로부터 부분계열 X_N을 복원하는 방법에 대하여 기술한다. M상의 모든 행과 열의 부분계열을 이루는 구성 기호는 X_N의 기호와 동일하다. 또한, 각 S_k의 첫 기호 s_N^k로 이루어지는 부분계열을 F_N이라 할 때, $F_N=f_1\ f_2 \cdots f_N\ f \in A$은 L_N의 요소 g를 사전식 순서로 정렬한 것과 같다. L_N과 F_N의 관계를 이용하면 각 s_k에 있어서의 모든 기호 s_k^i를 구할 수 있다. 이를 위해서는 L_N과 F_N의 각 기호의 쌍(g_i, f_j)에 대응하는 M상의 행의 위치 정보를 나타내는 벡터 $T[i]$가 필요하다. 이 $T[i]$는 $g_i=f_j$ $(1\le i, j\le N)$일 때, $T[i]=j$이다. 단, $g_i=f_j$를 만족하는 j가 복수 존재할 경우 최소한 j가 선택된 후 다음의 선택에서 그 f_j는 제외된다. 그 결과 $F[T[i]]=L[i]$의 관계가 성립한다. 여기서, $A[i]$는 계열 A의 i번째 요소를 나타낸다.

$L[i]$로부터 계산된 $T[i]$와 수신된 행 번호 k를 이용하여 원래의 부분계열 X_N을 재귀적 처리로 복호하게 된다. $L[i]$와 $F[i]$를 각각 M상의 $i-1$행의 마지막과 첫 기호라 할 때, 각 $i=1, \cdots, N$에 대하여

$$X[N+1-i]=L[T^i[k+1]]$$

의 관계가 성립한다. 여기서, $T^1[x]=x$, $T^{i+1}[x]=T[T^i[x]]$이다. 앞의 예제에서 구한 (L_N, k)를 이용하여 X_N을 복호하는 과정을 다음에 나타낸다.

(1) L_N을 사전식으로 정렬하여 F_N을 구한다.

$L_6=$"$cbdaab$" ⇨ $F_6=$"$aabbcd$"

(2) L_N과 F_N의 관계로부터 벡터 $T[i]$를 구한다.

i	:	[1 2 3 4 5 6]
$L[i]$	=	[c b d a a b]
$F[j]$	=	[a a b b c d]
j	:	[1 2 3 4 5 6]

⇩

$T[i]=[5\ 3\ 6\ 1\ 2\ 4]$ ⇨ $F[T[1]]=F[5]=L[1]=c$, $i=1$

(3) $L[i]$, $T[i]$와 k로부터 $X_N=x_1 \cdots x_N$을 복원한다.

$L[1, \cdots, 6]=[c, b, d, a, a, b]$
$T[1, \cdots, 6]=[5, 3, 6, 1, 2, 4]$, $k=1$

⇩

i	$X[N+1-i]=L[T^i[k+1]]$
1	$X[6]=L[T^1[2]]=L[2]=b$
2	$X[5]=L[T^2[2]]=L[T[T^1[2]]=L[T[2]]=L[3]=d$
3	$X[4]=L[T^3[2]]=L[T[T^2[2]]=L[T[3]]=L[6]=d$
4	$X[3]=L[T^4[2]]=L[T[T^3[2]]=L[T[6]]=L[4]=b$
5	$X[2]=L[T^5[2]]=L[T[T^4[2]]=L[T[4]]=L[1]=c$
6	$X[1]=L[T^6[2]]=L[T[T^5[2]]=L[T[1]]=L[5]=a$

따라서, $X[1, \cdots, 6]=$"$acabdb$"로 되어 원래의 부분계열 X_N이 복호됨을 알 수 있다.

한편, 순회정렬에 의한 부분계열 L은 각 S_k의 첫기호 s_1^k을 기준하여 정렬이 이루어지기 때문에

s_i^N이 취할 수 있는 기호는 어떤 확률분포 P에 의해 제한된다. 따라서, s_i^N의 연접으로 이루어지는 L은 s_i의 조건하에 생성되기 때문에 원래의 계열 X를 1차 마르코프 모델링하는 효과를 얻게 되므로 그 엔트로피가 상당히 감소된다.

3. 선두 이동법(MTF)

선두 이동법(Move To Front)은 q개의 서로 다른 요소로부터 구성되는 사전 $D=\{d_0, \cdots, d_{q-1}\}$ 중의 한 요소를 순차검색할 때 검색된 요소 d_i를 사전의 선두로 옮기고, 그 이후의 요소들은 1자리씩 뒤로 이동시켜 재나열하는 방법이다. 이것은 기억장치의 페이징 방법의 하나로 컴퓨터 시스템에 널리 사용되어 온 것이나, BSTW[5]와 Elias[6]에 의해 데이터 압축에 응용이 독립적으로 연구되어 MTF와 재귀순위(recency rank)방식으로 널리 알려지게 되었다. MTF는 자주 출현하는 요소일수록 선형 리스트로 나타내어지는 사전 D의 선두부분에 밀집되기 때문에 정수의 유니버샬 표현이나 엔트로피 부호화가 효과적으로 작용한다.

3.1 MTF 부호화

2.1절의 블럭정렬에 의해 얻어진 부분계열 L_N에 MTF를 적용한다. 먼저, 선형 리스트 Y를 정보원 기호 $\{a, b, c, d\}$가 사전식으로 등록되도록 초기화한다. 이 리스트 Y는 블럭정렬의 출력 L_N="*cbdaab*"에 따라 다음과 같이 갱신된다.

L_N	Y	R_N
$g_1=c$	$Y_1=\{a, b, \underline{c}, d\} \Rightarrow 2 \rightarrow r_1$	
$g_2=b$	$Y_2=\{c, a, \underline{b}, d\} \Rightarrow 2 \rightarrow r_2$	
$g_3=d$	$Y_3=\{b, c, a, \underline{d}\} \Rightarrow 3 \rightarrow r_3$	
$g_4=a$	$Y_4=\{d, b, c, \underline{a}\} \Rightarrow 3 \rightarrow r_4$	
$g_5=a$	$Y_5=\{\underline{a}, d, b, c\} \Rightarrow 0 \rightarrow r_5$	
$g_6=b$	$Y_6=\{a, d, \underline{b}, c\} \Rightarrow 2 \rightarrow r_6$	

이때, Y_i상의 g_i의 위치 $I=\{0, \cdots, m-1\}$를 차례로 연접한 길이 N의 새로운 부분계열 $R_N=r_1\ r_2 \cdots r_N$, $r_i \in I$가 MTF의 출력이 된다. 즉, L_6="*cbdaab*"의 MTF출력은 R_6="223302"로 변환된다.

3.2 MTF 복호화

MTF에 의해 변환된 R_N으로부터 순회정렬 계열 L_N을 복원하기 위해서는 부호화에서와 같이 선형리스트 Y를 r_i에 따라 갱신하면서 수행한다. 예를 들면, 앞의 MTF 부호화에서 얻어진 R_6="223302" 계열과 초기 리스트 $Y=[a, b, c, d]$가 주어지면 복호는 다음과 같이 이루어 진다.

R_N	Y	L_N
$r_1=2$	$Y_1=\{a, b, \underline{c}, d\} \Rightarrow c \rightarrow g_1$	
$r_2=2$	$Y_2=\{c, a, \underline{b}, d\} \Rightarrow b \rightarrow g_2$	
$r_3=3$	$Y_3=\{b, c, a, \underline{d}\} \Rightarrow d \rightarrow g_3$	
$r_4=3$	$Y_4=\{d, b, c, \underline{a}\} \Rightarrow a \rightarrow g_4$	
$r_5=0$	$Y_5=\{\underline{a}, d, b, c\} \Rightarrow a \rightarrow g_5$	
$r_6=2$	$Y_6=\{a, d, \underline{b}, c\} \Rightarrow b \rightarrow g_6$	

2.2절에서 밝힌 바와 같이 블럭정렬을 취하면 S_k의 첫기호 s_k^1의 조건하에 발생하기 쉬운 제한된 기호들이 한 곳으로 집중되게 된다. 따라서, MTF를 적용하면 인접기호들 간의 반복시간이 짧게 되어 위치정보 I는 0을 중심으로 하는 작은 숫자들이 높은 확률로 나타나는 기하분포를 이루게 된다. 이 계열 R에 대한 밀집도는 인접기호 간의 차분 절대치를 평균한 유사도에 의해 평가될 수 있다[7].

4. VFAC(Variable to Fixed Arithmetic Code)

VFAC는 일반 산술부호와 블럭부호를 결합시킨 형태로서 입력계열을 기호의 발생확률에 비례하게 가변길이의 부분열로 분해하여 고정길이의 부호어를 출력한다. VFAC는 일반 산술부호와는 달리 i.i.d(independently and identically distributed) 정보원에 대하여 전송오류로 인한 동기(synchronization)를 잃지 않기 때문에 치명적 오류가 발생하지 않으며, 오류의 영향이 1 부호어에 한정된다는 장점을 가진다. 그러나, 입력 정보원을 마르코프 모델로 확장할 경우, 이러한 장점은 소멸된다. 또한 정적 부호화에 필요한 발

생확률 표현을 위한 오버헤드가 지수적으로 급증하게 된다. 따라서, 마르코프 정보원에 대하여 오류제어 기능을 가지면서 오버헤드를 최소화시키는 효율 좋은 VFAC를 제안한다.

4.1 1차 마르코프 정보원에 대한 VFAC

VAFC가 1차 마르코프 정보원에 대해서 오류의 전파를 막아주는 블럭부호의 기능을 유지하도록 부분적으로 i.i.d 정보원의 발생확률을 이용한다. 즉, t번째 블럭의 첫 기호 x_{t1}은 앞의 기호와 독립적으로 발생하는 i.i.d 정보원에 의한 기호의 발생확률 $p(x_{t1})$을 이용하고, 첫 기호를 제외한 나머지 기호 $x_{tj}(j=2, 3, \cdots)$들은 앞의 기호 x_{tj-1}에 의존하여 발생하는 1차 마르코프 정보원에 의한 기호의 조건부 확률 $p(x_{tj} \mid x_{tj-1})$을 이용하여 부호화한다. 따라서 입력 기호간의 의존적인 상관성을 이용하지만 각 출력부호어는 독립적으로 된다. 이와 같이 정보원 모델을 혼합하여 부호화할 경우, 압축효율은 모든 입력기호를 마르코프 모델의 정보원으로 부호화하는 경우보다는 약간 떨어지지만 오류전파의 발생을 막을 수 있는 오류제어 기능을 갖게 된다.

i.i.d 정보원에 대한 입력기호의 집합 $A=\{a_1, a_2, \cdots, a_m\}$이고, 1차 마르코프 정보원에 대한 기호의 집합 $B=\{b_1, b_2, \cdots, b_m\}$라고 할 때, B의 부분집합 b_i는 현재 기호 $a_i \in A$ 다음에 발생하는 기호의 집합으로

$$b_i=\{a_{i1}, a_{i2}, \cdots, a_{iM_i}\}$$
$$a_{ij} \in A,\ 1 \le i \le m,\ 1 \le j \le M_i$$

이다. 집합 A에 있는 기호의 발생확률 $p(a_1)=p_1$, $p(a_2)=p_2$, $\cdots$, $p(a_m)=p_m$ 및 1차 마르코프 정보원 기호의 집합 B의 각 b_i에 있는 기호의 발생확률 $p(a_{i1})=p_{i1}$, $p(a_{i2})=p_{i2}$, $\cdots$, $p(a_{iM_i})=p_{iM_i}$은 각각 내림차순 정렬이며, 정보원 기호의 집합 A와 집합 B의 각 b_i에 대한 누적확률 C_l 및 C_{il}는 식(1)과 식(2)와 같다.

$$C_0=0,\ C_l=\sum_{k=1}^{l} p_k \quad \cdots\cdots (1)$$

$$C_{i0}=0,\ C_{il}=\sum_{k=1}^{l} p_{ik} \quad \cdots\cdots (2)$$

1차 마르코프 정보원에 대한 VFAC는 출력부호어의 집합을 m과 $M'_i(M'_i \le M_i)$개의 부분집합으로 완전분할한다. 즉, 분할 과정을 트리의 관점에서 보면, 트리의 근(root)은 i.i.d 모델에 의해 정보원 기호의 집합 A를 기준으로 m개의 자노드로 분할된다. 그러나, 중간노드는 1차 모르코프 모델에 의해 집합 B를 기준으로 부호어와 집합 크기 K가 $K \ge M_i$일 경우는 M_i의 자노드로, $K<M_i$일 경우는 M'_i개의 자노드로 분할된다. 출력 부호어의 집합은 출력 부호어의 고정길이 w에 의해 $W=\{0, 1, \cdots, 2^w-1\}$가 되며, 출력 부호어 집합의 크기 K의 초기값은 2^w-1이다.

출력 부호어의 집합 크기 K는 두 정보원 기호의 집합에 대한 발생확률 및 누적확률을 이용하여 식(3)에 의해 계산한다.

$$\begin{cases} K_l=\lceil \dfrac{p_l}{C_l} K+0.5 \rceil \text{ for } l=1, 2, \cdots, m \\ \quad : \text{i.i.d 정보원} \quad \cdots\cdots (3.a) \\ K_{il}=\lceil \dfrac{p_{il}}{C_{il}} K+0.5 \rceil \text{ for } l=1, 2, \cdots, M'_i \\ \quad : \text{1차 마르코프 정보원} \quad \cdots\cdots (3.b) \end{cases}$$

단, $\lceil x \rfloor$는 x보다 작은 최대 정수이다. 출력 부호어 집합의 재귀적 완전분할을 통하여 부분집합의 크기 $K<M_i$가 되면, 먼저 $1 \le M'_i \le M_i$을 만족하는 적절한 정보원 기호의 수 M'_i를 선정한다. 그리고 발생확률이 높은 기호순으로 $b'_i \subset b_i$인 새로운 정보원 기호의 집합 $b'_i=\{a_{i1}, a_{i2}, \cdots, a_{iM'_i}\}$을 설정한 다음 집합 b_i에 대하여 트리를 완전분할하며, $K=1$일 때, 부호어를 출력하고 재초기화하여 반복 수행한다. 그러나, 집합 b_i의 설정 후 입력되는 기호가 집합 b'_i에 존재하지 않는 경우는 분할을 중지하고, $K=1$일 때와 마찬가지로 부호어를 출력하고 재초기화한다. M'_i의 선정은 부호어 집합의 크기 K에 따라 식(4)와 같이 설정하며 새로운 정보원 기호의 집합 b'_i는 발생확률이 높은 기호순으로 M'_i개를 포함하도록 한다.

$$\begin{cases} M'_i=K \quad : 1<K \le \dfrac{M_i}{3} \\ M'_i=\left\lfloor \dfrac{M_i}{3} \right\rfloor : \dfrac{M_i}{3}<K \le \dfrac{M_i}{2} \end{cases}$$

$$\begin{cases} M_l' = \left\lceil \frac{M_l}{2} \right\rceil : \frac{M_l}{2} < K \le \frac{2M_l}{3} \\ M_l' = \left\lceil \frac{2M_l}{3} \right\rceil : \frac{2M_l}{3} < K \le M_l \end{cases} \quad \cdots\cdots (4)$$

따라서, VFAC는 아래와 같이 정의되는 정보원 기호수 q에 대하여 출력부호어의 집합을 완전분할한다.

$$q = m : \text{i.i.d정보원} \quad \cdots\cdots (5)$$

$$\left.\begin{array}{ll} q = M_l & \text{if } K \ge M_l \\ q = M_l' & \text{if } K < M_l \end{array}\right\} : \text{1차 마르코프 정보원}$$

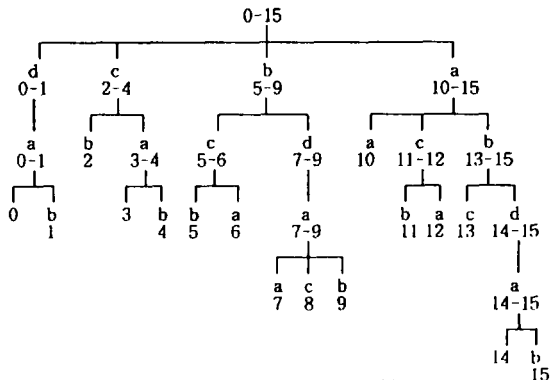
〈표 1〉 각 정보원에 대한 기호의 발생확률
〈Table 1〉 Symbol probability in each source

i.i.d 정보원				1차 마르코프 정보원					
집합	기호	순위 (l)	발생 확률	집 합		현재 기호	다음 기호	순위 (l)	발생 확률
A	a	1	0.4	B	b_1	a	b	1	0.5
							c	2	0.3
							a	3	0.2
	b	2	0.3		b_2	b	d	1	0.6
							c	2	0.4
	c	3	0.2		b_3	c	a	1	0.7
							b	2	0.3
	d	4	0.1		b_4	d	a	1	1.0

예를 들어, 정보원 기호의 집합 A와 B에 대한 기호의 발생확률이 〈표 1〉과 같고, 출력부호어의 길이가 $w=4$비트라고 하자. $m=4$, $M_1=3$, $M_2=2$, $M_3=2$, $M_4=1$이 되며, 초기 부호어 집합의 크기 $K=2^4$의 출력 부호어 집합 $W=\{0, 1, \cdots, 15\}$에 대하여 트리는 (그림 1)과 같이 분할된다. 트리는 발생확률이 낮은 기호부터 분할하므로 부분집합의 크기 K_l은 K_q, K_{q-1}, …, K_1의 순서로 계산한다.

트리의 레벨 $L=0$에서는 i.i.d 정보원의 발생확률에 비례하게 출력 부호어의 부분집합 W_l의 크기 K_l을 식(3.a)에 의해 구한다. 따라서, 초기 부호어의 집합 W는 $K_4=2$, $K_3=3$, $K_2=5$, $K_1=6$개의 부호어를 갖는 $m=4$개의 부분집합 W_4, W_3, W_2, W_1로 분할된다.

입력계열 X에서 최초의 기호 $x_1 \in A$가 'a'이면 첫 번째 부분집합 $W_1=\{10, 11, \cdots, 15\}$이 선택되어 $K=6$이 된다. $L=1$에서 부터는 1차 마르코프 정보원의 발생확률에 따라 K_l을 식 (3.b)에 의해 구하므로 입력기호 'a' 다음에 발생하는 기호의 집합 $b_1=\{b, c, a\}$에 대하여 $K_3=1$, $K_2=2$, $K_1=3$의 부호어수를 갖는 $M_1=3$개의 부분집합으로 분할된다. 두 번째의 기호 x_2='b'이면 다시 첫 번째 부분집합 $W_1=\{13, 14, 15\}$인 $K=3$이 되고, 집합 $b_2=\{d, c\}$를 기준으로 $K_2=1$, $K_1=2$개의 부호어를 갖는 $M_2=2$개의 부분집합으로 분할된다. 입력기호 x_3이 'd'라면 $W_1=\{14, 15\}$인 $K=2$가 되고, 집합 $b_4=\{a\}$로 $M_4=1$이므로 기호 x_4는 당연히 'a'가 되므로 x_4에 대해서는 부호어의 집합을 분할할 필요가 없게 된다. 따라서 $K=2$에서 $b_1=\{b, c, a\}$에 대하여 입력기호 x_5를 처리하게 된다. 이 때 $K<M_1$이므로 식 (4)의 M_l'의 선정기준에 따라 $M_1'=1$이 되며, 새로운 정보원 기호의 집합 $b_1'=\{b\}$가 된다. 기호 x_5가 'b'이면, 부호어 15를 출력하고 그렇지 않으면 부호어 14를 출력하여 재초기화 한 후, $L=0$에서 다시 부호화를 시작한다. 따라서 $L=0$에서는 항상 $K \ge m$을 만족하므로 집합 A의 정보원 기호수 m에 대하여 트리를 완전분할하고, $L \ge 1$에서는 집합 b_l의 정보원 기호수 M_l에 대하여 트리를 완전분할한다. 이때 $K<M_l$가 되면 먼저 $1 \le M_l' < M_l$을 만족하는 적절한 정보원 기호수 M_l'를 선정한 후, 새로 선정된 정보원 기호의 집합 b_l'를 선정한 후, 집합 b_l'에 대하여 트리를 완전분할한다.



(그림 1) VFAC의 트리 분할
(Fig. 1) Tree splitting of VFAC

VFAC는 식(3)에 의해 K_l을 구하여 [알고리즘1]과 같이 GQ(Good Quantizer)을 이용하여 부호화한다. GQ방식에 의한 부호화는 발생확률이 가장 낮은 기호부터 높은 기호순으로 부호어의 집합이 분할되며, 이 때 각 기호에 대해 남아있는 부호어수($K=K-K_j$)를 기록한다. 현재의 나머지 확률(누적확률)에 대한 현재 기호의 발생확률(p_j/C_j 또는 p_{ij}/C_{ij})을 이용하므로 1개의 부호어에 여러개의 정보원 기호가 할당되는 충돌현상을 방지할 수 있다. 그러나, GQ방식은 정보원의 기호수가 많아지게 되면 처리속도가 크게 떨어지므로 VAFC를 고속화하기 위한 FA(Fast Quantizer)방식을 아래와 같이 제안한다.

VFAC의 FQ방식은 발생기호의 누적확률만을 이용하여 부호어의 집합을 분할한다. FQ방식에서는 충돌현상을 막기 위하여 현재 부분집합의 크기 K에서 정보원 기호수 만큼을 감산하여 각 부분집합에 1개씩의 부호어를 할당($K-q$)한 후, i.i.d 정보원 및 1차 마르코프 정보원의 각각에 따라 식(6)과 식(7)에 의해 누적 부분집합의 크기 CK_l 또는 CK_{il}을 구하여 부분집합의 크기 K_l을 구한다. K_q, K_{q-1}, …, K_{l+1}의 부분집합 크기를 모두 합한 값에 대한 누적값이 [알고리즘2]에서 low에 해당한다. FQ방식은 각 입력기호에 대한 반복연산을 제거함으로써 처리속도가 상당히 향상되며, 정보원 기호수가 큰 데이터에 대해서도 GQ방식과 동등한 압축효율을 유지한다.

$$\begin{cases} CK_l = l + \lceil (K-q) * \dfrac{C_l}{C_q} \rceil \\ K = CK_l - CK_{l+1} \end{cases} \quad \cdots\cdots\cdots\cdots (6)$$

$$\begin{cases} CK_l = l + \lceil (K-q) * \dfrac{C_{il}}{C_{iq}} \rceil \\ K = CK_{il} - CK_{il-1} \end{cases} \quad \cdots\cdots\cdots\cdots (7)$$

[알고리즘1] GQ방식에 의한 K_l의 계산

```
1 : for(j=q ; j>l ; j--) {
2 :       if(L==0)
3 :           K_j = ⌈ (p_j / C_j) K + 0.5 ⌉ ;
4 :       else
5 :           K_j = ⌈ (p_ij / C_ij) K + 0.5 ⌉ ;
6 :       if(K_j==0)  K_j = 1 ;
7 :       K = K - K_j ;
8 : }
```

복호는 부호화와 동일하게 K_l을 구하여 식(8)을 만족하는 정보원 기호 a_l을 구한다. 이 때, 기호 a_l을 고속으로 찾기 위해 정보원 기호의 집합에 대해 이진탐색을 수행한다.

$$\text{low} + \sum_{j=l+1}^{q} K_j \le \text{Codeword} \le \text{low} \sum_{j=l}^{q} K_j \quad \cdots(8)$$

[알고리즘2] VAFC의 부호화

```
1 : flag=OFF ; L=0 ;
2 : while (1) {
3 :   if(lag==OFF)
4 :     if((l=getinput())==EOF) {
5 :       Output code(low) ; doeof(low, K) ; }
6 :   if(L==0) {          /* i.i.d */
7 :     K=2^u ; low=0 ;
8 :     Compute K_m, K_{m-1}, ..., K_{l+1}, K_l ;
9 :     low=low+ Σ_{j=l+1}^{m} K_j ; K=K_j ; L++ ; }
10 :  else if(K>M_j) {/* First order Markov */
11 :    Compute K_Mi, K_{Mi-1}, ..., K_{l+1}, K_l ;
12 :    low=low+ Σ_{j=l+1}^{Mi} K_j ; K=K_j ; L++ ; }
13 :  else if(K>1)1) {
14 :    K--; low++ ;
15 :    Compute M_i' ;
16 :    if(l≤M_i') {
17 :      Compute K_Mi, K_{Mi-1}, ..., K_{l+1}, K_l ;
18 :      low=low+ Σ_{j=l+1}^{Mi'} K_j ; K=K_j ; L++ ; }
19 :    else { flag=ON ; low--; K=0; L--; }
20 :  }
21 :  if(K≤1) {Output code(low) ; L=0 ; }
22 : }
```

4.2 발생확률의 표현

VFAC는 각 입력기호의 발생확률에 관한 헤드정보를 필요로 한다. 따라서, VFAC에 의한

부호어의 선두에 각 입력기호의 존재 여부에 관한 비트와 입력기호의 출현회수에 관한 정보가 포함된다.

입력기호의 존재여부를 나타내기 위한 정보는 $|A|=m$인 정보원의 경우, i.i.d 정보원의 m 비트에 대하여 n차 마르코프 정보원에서는 m^{n+1} 비트로 증가한다. 따라서, 1차 마르코프 정보원의 경우 m^2비트의 존재여부 비트가 필요하게 된다. 허프만 부호어 테이블을 효과적으로 나타내는 방법이 T. Kawamura 등에 의해 고안되어 있

대응 / 표현	0번째 ↓	ab에 대응 (24930번째) ↓		by에 대응 (25209번째) ↓		cx에 대응 (25464번째) ↓			xa에 대응 (30817번째) ↓		65535 번째 ↓	단계
2진수	0^ 24928	0010	0^ 276	0100		0^ 252	1000		0^ 5348	0100	0^ 34716	(1)
BGC	0^ 24928	0011	0^ 276	0110		0^ 252	1100		0^ 5348	0110	0^ 34716	(2)
16진수	0^ 6232	3	0^ 69	6		0^ 63 ↓	C		0^ 1337	6	0^ 8679	(3)
	0^ 6232	3000	0^ 64	0060		0^ 60	00C0		0^ 1336	6000	0^ 8676	
BGC	0^ 6232	1100	0^ 64	0011		0^ 60	0011		0^ 1336	1100	0^ 8676	(4)
16진수	0^ 1558	C	0^ 16	3		0^ 15 ↓	3		0^ 334	C	0^ 2169	(5)
	0^ 1556	00C0	0^ 12	0003	0000	0^ 8	0003	0000	0^ 328	00C0	0^ 2168	
BGC	0^ 1556	0011	0^ 12	0001	1000	0^ 8	0001	1000	0^ 328	0011	0^ 2168	(6)
16진수	0^ 389	3	000	1	8	00 ↓	1	8	0^ 82	3	0^ 542	(7)
	0^ 388	0300		0180			0180		0^ 80	0300	0^ 540	
BGC	0^ 388	0110		0101			0101		0^ 80	0110	0^ 540	(8)
16진수	0^ 97	6		5		↓	5		0^ 20	6	0^ 135	(9)
	0^ 96	0655						0000	0^ 16	6000	0^ 132	
BGC	0^ 96	0100						1000	0^ 16	1100	0^ 132	(10)
16진수	0^ 24	4				↓		8	0^ 4	C	0^ 33	(11)
	0^ 24	4800								00C0	0^ 32	
BGC	0^ 24	1010								0011	0^ 32	(12)
16진수	0^ 6	A				↓				3	0^ 8	(13)
	0^ 4	00A3								0000	0^ 4	
BGC	0^ 4	0010								1000	0^ 4	(14)
16진수	0	2				↓				8	0	(15)
	0280											
BGC	0101											(16)
16진수	5											(17)

* 0^ m : m개의 0, BGC : Binary to Gray Code

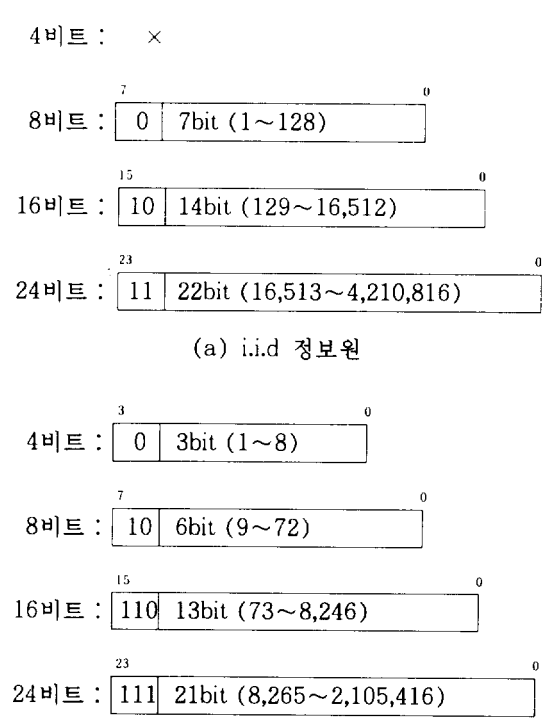
(그림 2) 존재여부 비트 압축 과정
(Fig. 2) Bit compression process

다[8]. i.i.d 정보원에 대하여 고안된 이들의 방법(이하, KES 방식)을 변형하여 1차 마르코프 정보원에 적용한다. KES 방식은 $m=256$인 ASC II 입력계열에서 기호의 존재여부에 따라 1/0으로 표시된 2진계열에 대하여 그레이 변환을 취하여 16진수 표현을 반복수행한다. 따라서, KES 방식은 그레이 변환을 1번만 수행하지만, 제안하는 방식은 KES 방식에 의해 생성된 16진수 표현에 대해 그레이 변환을 반복수행하게 된다. 1차 마르코프 정보원의 경우, 각 기호의 존재 위치는 [앞 기호×256+현재기호]번째 비트로 만약, 기호 'c'(99) 다음에 'a'(97)가 입력되는 경우가 있다면, 99×256+97=25,441번째 비트가 1이 되고, 그렇지 않으면 0이 된다. 예를 들어 입력계열 X="cxaby"에 대한 존재여부 비트는 (그림 2)와 같이 압축된다.

(그림 2)의 단계(1)에 표시된 2진 계열에 대하여 BGC(Binary to Gray Code)로 변환한 결과가 단계(2)이며, 그것의 16진수 표현이 단계(3)이다. 단계(3)에서 0이 아닌 값을 1로, 0인 값을 0으로 하여 BGC 변환을 취한 것이 단계(4)이며, 다시 4비트씩 묶어서 16진수로 표현한 결과가 단계(5)이다. 이것을 재귀적으로 반복수행하여 마지막 단계(17)이 되면 16진수 5가 남게 된다. 최종적으로 기억해야 할 정보는 16진수 표현의 단계에서 0이 아닌 값들을 최종 단계로부터 역순으로 연접한 것이다. 따라서, 1차 마르코프 정보원의 65,536비트의 존재여부 비트는 「5,28,A3,48C,6556,318183,C33C,36C6」의 104비트로 대폭 압축된다.

복호는 압축된 정보 104비트를 이용하여 역순으로 이루어진다. 먼저 압축된 정보「528A348C6556318183C33C36C6」의 선두 5에 대한 BCD 표현 [0101]의 GBC(Gray to Binary Code) [0110]을 구한다. [0110]에서 0은 4자리의 16진수 표현 [0000]으로 대체되고, 1에 의해 압축된 정보에서 각각에 대응하는 2와 8을 추출한다. 즉, [0110]이 [0000 0010 1000 0000]으로 신장된다. 이 값은 GBC 변환에 의해 [0000 0011 0000 0000]이 되고, 다시 0은 4자리의 16진수 표현으로, 1은 압축된 정보에서 새로운 값을 추출하여 재귀적으로 반복수행하면 최종적으로 단계(1)의 65,536비트가 복호된다.

다음으로 입력기호의 출현회수에 관한 정보는 i.i.d 정보원의 경우 (그림 3(a))와 같이 3가지의 형태로 나타내는 반면, 1차 마르코프 정보원의 경우 최대 출현회수와 최소 출현회수 사이에 격심한 차이가 존재하므로 (그림 3(b))와 같이 4가지의 형태로 나누어 출현회수를 표현한다. 따라서, 최대 42,180,816($2^{22}+2^{14}+2^{7}$) 및 2,105,416($2^{21}+2^{13}+2^{6}+2^{3}$)까지의 출현회수를 표현할 수 있으므로 각각의 정보원에 대하여 충분히 긴 데이터에도 대응할 수 있다.



(a) i.i.d 정보원

(b) 1차 마르코프 정보원

(그림 3) 출현회수의 부호화 형식
(Fig. 3) Coding format of symbol occurrence

5. 성능평가 및 고찰

블럭정렬과 VFAC에 의한 제안 방식의 성능을 평가하기 위하여 컴퓨터 시뮬레이션에 의한 압축실험을 실시하였다. 입력 데이터는 T.C.Bell 등에 의해 작성된 "Calgary Compression Corpus [1]" 가운데 다양한 형식의 7개의 데이터를 임의로 선정하였다. 압축율은 입력길이에 대한 출

력길이의 비 ρ로 나타내었다. 또한, 시뮬레이션은 UNIX 워크스테이션[Axil-220 : 50MHz micro SPARC : 59.1 MIPS] 상에서 C언어(GNU GCC Ver 2.6.3)로 구현하였다.

5.1 정규화 엔트로피의 비교

원래의 입력계열 X, 순회정렬된 계열 L, MTF에 의한 계열 M, 그리고 블럭정렬 후 MTF화된 계열 R에 대한 압축 한계를 나타내는 엔트로피를 $NH_i(X)=H_i(X)/\log_2 m$로 정규화하여 〈표 2〉에 나타낸다. 여기서, $H_i(X)$는 계열 X의 i차 마르코프 정보원 엔트로피이다.

원계열 X에 대한 $NH_0(X)$와 블럭정렬된 계열 L의 $NH_0(L)$은 동일한 값이다. 이것은 i.i.d 정보원으로 간주하였을 때 각 기호의 위치만이 변경되었을 뿐 블럭정렬된 계열의 발생빈도는 원래의 것과 같기 때문이다. 그러나, 1차 마르코프 정보원으로 확장하여 측정한 정규화 엔트로피는 모든 계열에 대하여 줄어듬을 알 수 있다. 특히, 2장에서 밝힌 바와 같이 계열 L에 있어서 $NH_0(L)$에 비해 $NH_1(L)$의 평균값이 33.2% 줄어들어 $NH_0(X)$에 대한 $NH_1(X)$의 감소율 20.7%보다 크기 때문에 블럭정렬의 효과를 알 수 있다. 또한, i.i.d 정보원으로 처리할 경우, 원래의 계열 X의 $NH_0(X)$보다 순회정렬 후 MTF를 취한 계열 R의 $NH_0(R)$이 30.6% 정도의 삭감을 보이고 있다. 반면, MTF만 취한 계열 M에 대한 $NH_0(M)$은 $NH_0(X)$에 비해 증가하는 경향이

〈표 2〉 정규화 엔트로피의 비교
〈Table 2〉 Comparison of normalized entropy

$NH_i=[H_i/\log m]$

data [byte]	$NH_0(X)$	$NH_0(L)$	$NH_0(R)$	$NH_1(X)$	$NH_1(L)$	$NH_1(R)$	$NH_0(M)$
bib [11,261]	0.650	0.650	0.285	0.420	0.284	0.256	0.701
bookl [768,771]	0.565	0.565	0.354	0.448	0.373	0.316	0.618
geo [102,400]	0.705	0.706	0.668	0.533	0.482	0.526	0.685
obj2 [246,814]	0.782	0.782	0.344	0.484	0.322	0.304	0.769
paper1 [53,161]	0.622	0.622	0.335	0.455	0.341	0.310	0.653
progc [39,611]	0.649	0.649	0.336	0.450	0.335	0.308	0.687
trans [93,695]	0.691	0.691	0.203	0.419	0.201	0.191	0.685
avg. [202,245.3]	0.666	0.666	0.360	0.459	0.334	0.316	0.685

〈표 3〉 원계열 X에 대한 VF형 산술부호의 압축율
〈Table 3〉 Compression ratio for original sequence X

[ρ=output/input]

data [byte]	i.i.d 정보원					1차 마르코프 정보원				
	AB-coding	BAC		VFAC		AB-coding	BAC		VFAC	
	BCM	FQ	GQ	FQ	GQ	BMC	FQ	GQ	FQ	GQ
bib [111,261]	.702	.695	.689	.675	.673	.486	.506	.504	.478	.475
bookl [768,771]	.606	.614	.608	.588	.585	.488	.516	.514	.481	.479
geo [102,400]	.799	.779	.763	.750	.746	.752	.785	.741	.729	.727
obj2 [246,814]	.870	.867	.853	.825	.822	.609	.650	.598	.599	.598
paper1 [53,161]	.673	.678	.669	.650	.646	.540	.564	.564	.530	.527
progc [39,611]	.709	.700	.700	.678	.676	.554	.580	.577	.544	.543
trans [93,695]	.754	.746	.735	.720	.718	.499	.524	.520	.491	.489
avg. [202,245.3]	.730	.726	.720	.698	.695	.561	.589	.534	.550	.548

있기 때문에 어떻게 정보원 분해를 수행할 것인가가 MTF만으로 부호화했을 때의 관건이 된다.

5.2 VF형 산술부호의 압축율 및 처리속도

오류제어 기능을 갖는 VFAC의 성능을 평가하기 위하여 기존에 제안된 AB-coding[2] 및 BAC[3]과 비교한다. 먼저, 3가지 부호화 방식을 원래의 입력계열 X를 i.i.d 정보원 및 1차 마르코프 정보원에 대하여 적용한다. AB-coding은 기본방식(BCM : Basic Coding Method)에 의한 부호화, BAC와 VFAC는 각각 GQ와 FQ방식에 의한 부호화의 압축율 및 처리속도를 〈표 3〉과 〈표 4〉에 표시하였다.

〈표 3〉에서 보는 바와 같이 제안하는 VFAC의 압축율은 i.i.d 정보원의 경우, AB_coding에 비해 약 3~3.5%, BAC 보다는 약 2.5~2.8% 향상되었고, 1차 마르코프 정보원에 대해서는 각각 약 1.1~1.3%, 2.6~3.9% 향상되었다. 또한, BAC의 경우 GQ방식과 FQ방식 간의 차이가 큰데 비해 제안한 VFAC에서는 거의 동등한 압축율을 유지함을 알 수 있다.

처리속도는 〈표 4〉에 나타낸 바와 같이 i.i.d 및 1차 마르코프 정보원에 대해서 VFAC의 FQ방식이 다른 방식에 비해 빠른 처리속도로 수행됨을 알 수 있다. GQ방식은 기호수에 대한 반복계산으로 처리속도가 느린 반면, FQ방식은 누적확률을 이용하므로 평균 10배 이상 고속화된다. AB-coding 및 VFAC의 FQ방식은 i.i.d 정보원

〈표 4〉 원계열 X에 대한 VF형 산술부호의 압축율
〈Table 3〉 Compression of processing time

단위 : sec.

data [byte]	i.i.d 정보원					1차 마르코프 정보원				
	AB-coding	BAC		VFAC		AB-coding	BAC		VFAC	
	BCM	FQ	GQ	FQ	GQ	BCM	FQ	GQ	FQ	GQ
bib	3.0	8.0	45.9	3.0	30.9	100.0	5.3	31.6	4.6	25.6
book1	17.4	58.1	338.5	17.7	228.8	78.2	32.4	290.2	26.2	233.7
geo	4.3	18.6	132.0	4.5	90.7	30.9	8.9	108.6	10.0	88.4
obj2	10.5	44.8	318.9	11.1	216.7	64.6	19.5	213.9	18.3	198.5
paper1	1.4	4.4	26.6	1.4	18.2	18.2	2.6	12.5	2.2	5.4
progc	1.2	3.3	18.5	1.1	12.6	4.4	2.2	14.5	1.8	11.8
trans	2.8	7.9	45.9	2.8	31.6	9.8	4.8	32.2	4.2	26.3
avg.	5.8	20.7	132.3	5.9	89.9	30.9	10.8	100.5	9.6	84.2

〈표 5〉 계열 X와 블럭정렬 계열 R에 대한 압축율 비교
〈Table 5〉 Compression ratio for each sequence

[ρ=output/input]

data	VFAC_GQ				overhead(byte)			
	i.i.d 정보원		1차 마르코프 정보원		i.i.d 정보원		1차 마르코프 정보원	
	X	R	X	R	X	R	X	R
bib	.672	.306	.455	.275	155	151	2,277	2,042
book1	.686	.380	.475	.350	170	172	2,830	2,584
geo	.742	.716	.588	.582	387	385	14,257	20,689
obj2	.820	.372	.543	.335	451	357	13,550	14,024
paper1	.644	.359	.486	.332	144	144	2,160	1,838
progc	.672	.358	.487	.330	145	149	2,245	2,019
trans	.716	.223	.459	.209	178	158	2,878	1,937
avg.	.693	.388	.499	.345	232.9	216.6	5,742.4	6,447.6

에 비해 1차 마르코프 정보원의 처리속도는 각각 평균 5배, 2배 정도 떨어지나 그 외의 부호화는 1차 마르코프 정보원의 경우가 1~2배 정도 더 빠른 처리속도로 실행된다.

5.3 블럭정렬 계열에 대한 압축율

다음은 원래의 입력계열 X와 블럭정렬 후 MTF된 계열 R에 대해 VFAC 부호화의 성능을 오버헤드 정보와 더불어 비교 평가한다. 두 정보원에 대하여 계열 R에 대한 각각의 압축율은 계열 X에 대한 압축율에 비해 평균 30.5% 및 15.4% 향상되었다. 발생확률을 표현하기 위한 오버헤드는 i.i.d 정보원의 경우 계열 X에 비해 계열 R은 약 16.3 byte 감소한 반면, 1차 마르코프 정보원의 경우 약 705.2 byte 증가하였다. i.i.d 정보원의 경우 오버헤드는 압축 데이터의 0.12%, 0.1%로 작은 비율을 차지하지만, 1차 마르코프 정보원의 경우는 2.8%, 3.2%로 i.i.d 정보원에 비해 증가하였다. 따라서, bib 및 book1과 같이 데이터의 길이가 길수록 오버헤드의 비율은 낮아지나, 짧은 데이터에 대해서는 압축율의 향상을 위해 마르코프 정보원의 차수를 증가시킬 수 없는 제약조건이 된다.

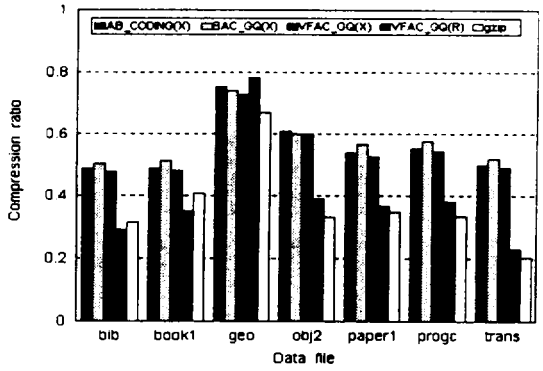
다음으로 텍스트 데이터 압축법으로 널리 이용되고 있는 gzip[9]와 LHa[10]의 결과와 1차 마르코프 정보원에 대하여 블럭정렬 후 MTF의 처리에 의한 계열 R에 VFAC의 부호화를 적용한 성능(오버헤드 정보 포함)을 비교하였다.

〈표 6〉 실용 유틸리티와 압축율 비교
〈Table 6〉 Comparison of compression ratio with utilities
[ρ=output/input]

data	gzip	LHz	VFAC_GQ
bib	.315	.366	.293
book1	.407	.441	.353
geo	.668	.670	.784
obj2	.330	.344	.392
paper1	.349	.371	.367
progc	.335	.353	.381
trans	.202	.241	.230
avg.	.372	.398	.400

〈표 6〉의 결과로부터 오버헤드의 비율이 작은 bib나 book1의 데이터에 대해서는 VFAC의 부호화가 우수하지만, geo와 obj2와 같이 오버헤드의 비율이 큰 데이터의 경우는 효율이 약간 떨어짐을 알 수 있다. 그러나, 제안하는 VFAC부호는 압축의 기능뿐만 아니라 전송로상에서 발생하는 오류를 해당 블럭에 제한시킬 수 있기 때문에 기존의 압축 유틸리티와는 다른 응용이 기대된다. 물론, 마르코프 모델의 차수를 증가시켜 압축율의 향상을 추구할 수 있지만, 발생확률 표현을 위한 오버헤드도 증가되어 상반되는 데이터 압축의 근본문제에 봉착하게 되므로 응용에 따른 사용자의 선택이 요구된다.

끝으로, VFAC와 VF형 산술부호 및 실용 유틸리티 gzip의 성능을 함께 비교하였다. 〈그림 4〉에 나타난 바와 같이 원계열 X에 대해 VF형 산술부호 AB-coding 및 BAC에 비해 VFAC의 성능이 우수하며, 원계열 X를 블럭정렬 및 MTF의 전처리를 행한 계열 R에 대한 제안방식의 압축율은 상당히 향상됨을 알 수 있다. 또한 bib 및 book1과 같이 길이가 긴 데이터에 대해서 실용 유틸리티 gzip에 비해 VFAC의 성능이 우수하였다. 그러나, 지진파의 관측 데이터인 32비트 단위의 수치 데이터 geo에 대해서는 MTF가 부적합함을 알 수 있다.



〈그림 4〉 압축율 비교
〈Fig. 4〉 Comparison of Compression ratio

6. 결 론

VF형 산술부호는 압축효율이 뛰어난 산술부호와 오류전파를 블럭단위로 제한시킬 수 있는 블럭부호를 결합시킨 방식이다. 기존의 데이터 압축 기술은 효율성을 중시한 압축율과 처리속도의 관점에서 연구되어 왔으나, 최근 효율뿐만 아니라 신뢰성의 기능을 동시에 고려하는 자기 동기성 부호에 관심이 집중되고 있다. 이러한 요구를 충족시키기 위한 가변길이 입력에 대한 고정길이 출력 형태의 산술부호에 대하여 고찰하였다. 특히, i.i.d 정보원에 대해 고안되어 있는 기존의 VF형 산술부호를 마르코프 정보원으로 확장함에 따른 문제점을 해결한 VFAC 방식을 제안하였다. 이때, 정적 부호화를 위한 발생기호의 확률 표현법으로써 그레이 변환의 반복적 수행에 의한 방법을 제시하였다. 또한, 압축율을 향상시키기 위해 블럭정렬과 MTF기법을 결합시킴으로써 기존의 실용 유틸리티와 대등한 압축율을 달성하였으며, 오류제어의 기능을 갖는 또 다른 응용이 기대된다.

제안방식을 화상 부호화에 응용함에 있어서 저작권 보호의 정보를 내장하는 화상 심층 암호와 결합시킴으로써 압축, 암호 및 오류제어의 세 기능을 갖는 응용이 향후의 과제이다.

참 고 문 헌

[1] 박지환, "문서 데이터 압축 알고리즘 입문," p.289, 성안당, 1995.

[2] J. Teuholar, T. Raita, "Arithmetic Coding into Fixed-Length Codewords," IEEE Trans. Inform. Theory, Vol.40, No.1, pp. 219-223, Jan.1994.

[3] G. Charles, Jr. Boncelet, "Block Arithmetic Coding for Source Compression," IEEE Trans. Inform. Theory, Vol.39, No. 5, pp.1546-1553, Sept.1993.

[4] M. Burrows, D. J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm," SRC Research Report 124 on DEC, Ma.y 1994.

[5] J. L. Bentley et al, "A Locally Adaptive Data Compression Scheme," Communications of ACM, Vol.29, No.4, pp.320-330, April. 1986.

[6] P. Elias, "Interval and Recency Rank Source Coding" IEEE Trans., Vol.IT-33, No.1, pp.3-10, Jan. 1987.

[7] 朴志煥, 今井秀樹, "Block-sortingデータ壓縮に關する考察," 電子情報通信學會 技術報告 IT94-99, pp.43-48, Mar. 1995. (in Japanese).

[8] 河村知行, 江口賢和, 重村哲至, "ハフマンコード表の壓縮とその應用," 情報處理學會論文誌, pp.267-271, Feb. 1994. (in Japanese)

[9] J. Cailly et al, "GZIP, Version 1.2.4," Anonymous ftp from prep.ai.mit.edu : /pub/gnu/gzip-1.2.4.tar.gz

[10] M. Oki, "LHa for UNIX V1.00", Anonymous ftp from garbo.uwasa.fi : /unix/arcers/lha-1.00.tar.Z

이 진 호

1974년 영남대학교 전자공학과 졸업(공학사)
1981년 영남대학교 전자공학과 (공학석사)
1991년 영남대학교 전자공학과 박사과정 수료
1979년~현재 경북산업대학교 전자계산학과 교수
관심분야 : 데이터 압축, 프로그래밍 언어, 객체지향 시스템

조 숙 희

1993년 부산수산대학교 전자계산학과(이학사)
1994년~현재 부산수산대학교 전자계산학과 석사과정
1993년~94년 부산수산대학교 전자계산학과 조교
관심 분야 : 데이터 압축, 오류제어 부호, 화상처리 등

박 지 환

1984년 경희대학교 전자공학과 졸업(공학사)
1987년 日本國立電氣通信大學 情報工學科 修了(工學修士)
1990년 日本橫浜國立大學 電子情報工學科 修了(工學博士)
1990년～현재 부산수산대학교 전자계산학과 조교수
1994년～95년 日本東京大學生産技術研究所 客員研究員
관심 분야 : 데이터 압축, 암호학 응용, 오류제어 부호, 화상처리 등

강 병 욱

1970년 영남대학교 전기공학과 (공학사)
1977년 영남대학교 전자공학과 (공학석사)
1994년 경북대학교 전자공학과 (공학박사)
1979년～현재 영남대학교 전산공학과 교수
관심분야 : 소프트웨어 공학, 프로그래밍 언어, 데이터 압축