通信情報保護學會誌
第2卷, 第2號, 1992. 6

Ziv-Lempel 부호와 그 응용에 관한 고찰†

A Study on Ziv-Lempel Codes and Its Applications

박 지 환*

1. 서 론

Ziv-Lempel 부호는 정보이론가인 J. Ziv와 컴퓨터 과학자인 A. Lempel의 공동작업에 의해 이루어진 결과의 일부로 Kolmogorov의 계산론적 정보량[1]에 그 바탕을 두고 있다. 계산론적 기법은 개개의 기호열(individual sequences)의 랜덤성에만 의존하고 정보원의 통계적 성질에 의존하지 않는 특징을 갖는다. 따라서 Run-length부호[2]와 같은 조합론적 기법이나 Huffman부호[3]와 같은 확률론적 정보량에 의한 통계적 기법과 달리 정보원 엔트로피의 측정이나 관측을 미리 행할 필요가 없는 만능성(universality)을 갖는다. 더우기, 이 기법에 의해 달성되는 평균부호의 길이는 대상으로 하는 입력 기호열의 길이를 무한히 길게 했을 때 이론적 한계인 엔트로피에 반드시 근접하는 점근적 최량성(asymptotically optimal)을 갖는다. 이 논문에서는 Ziv-Lempel의 계산론적 정보량과 그에 기초한 유니버살 데이타 압축에 널리 사용되는 Ziv-Lempel 부호와 그 응용에 대하여 알아본다.

2. Lempel-Ziv의 계산론적 정보량

이 정보량(measure of information)은 1976년 Lempel과 Ziv가 기호의 복제(copy)를 기본 조작으로 한 기호열 생성과정의 계산량을 도입한 양이다[4]. 유한열 R=12323232에 대해 P=123, Q=23232일때, R=P·Q이라 하면 기호열 Q는 기호열 P의 부분열인 23의 복제에 의해 재생된다. 이하 X·Y는 기호열 X와 Y의 연결(concatenation)을 나타낸다. 즉, P를 먼저 복제하고 이어서 시점 2의 P의 부분열 23을 반복하여 R의 길이만큼 복사하면 유한 기호열 R을 얻을 수 있다. 일반적으로 R=P·Q에서 Q가 P로부터의 복제이면 기호열 R은 P로부터 재생가능(reproducible)이라 하고 P ⇒ P·Q로 표기한다. 예를들면 aab ⇒ aab·abab로 표기된다.

한편, 기호열 : R=12323234에 대해 P=123, Q=23234, R=P·Q일때 R은 P로부터 재생가능하지 않지만 R(또는 Q)의 가장 오른쪽 기호인 4를 제외한 기호열 $\hat{R}$=1232323은 P의 부분열 23의 2회 반복으로 복제된다. 이와같을때 R은 P로부터 생성가능(producible)이라 하고 P → P·Q로 표기한다. 예를들면 aab → aab·ababb로 된다. 따라서, R은 복제에 사용하는 기호열 P의 부분열의 시점p, 복제하는 기호의 갯수 ℓ 및 가장 오른쪽 기호로 이루어지는 세가지 파라메타에 의해 생성 가능하다.

α개의 기호로 구성되는 정보원 알파벳 A={0, 1, …, α−1}, α=|A|에 대해 A*는 A상의 기호의 유한열 전체의 집합이다. 길이 $\ell(X)=n$인 A상의 기호열

† 이 논문은 1991년도 교육부 지원 한국학술진흥재단의 자유공모(지방대학육성)과제 학술연구조성비에 의하여 연구되었음

* 부산수산대학교 전자계산학과

$X=x_1 \cdots x_n \in A^*$의 위치 i에서 j까지의 부분열을 X(i, j)라 할 때

$$X(i, j)=x_i x_{i+1} \cdots x_j, \quad (i \le j)$$
$$X(i, j)=\lambda, \quad (i>j)$$

로 표현된다. 여기서 λ는 길이 0의 공열(null string)이다.

$0 \le \forall j \le \ell(X)$에 대해, X(1, j)는 X의 어두(prefix)이고 X(j, ℓ(X))는 X의 어미(postfix)이다. $j<\ell(X)$일때 X(i, j)를 proper prefix라 한다. 또, x가 y의 prefix일때 $x \subseteq y$로 표기한다. 기호열 $X=x_1 \cdots x_n$에 대해, d는 $Xd=x_1 \cdots x_{n-1}$를 만족하는 함수이다. 기호열 X의 부분열 전체의 집합을 $\nu(X)$라 하면 예를들어 $\nu(01)=\{\lambda, 0, 1, 01\} \subset A^*$이다.

[정의 1] 재생가능(reproducibility)

$R, P \in A^*, P \subseteq R, R=P \cdot Q$에 대해 $\nu(Rd) \supseteq Q$일때, R은 P로부터 재생가능이라하고 $P \Rightarrow R$로 표기한다. 복제에 이용하는 기호열 P의 부분열의 시점위치 p를 재생포인터라 하면 $Q=R(p, \ell(Q+p-1)$이다.

[정의 2] 생성가능(producibility)

λ이 아닌 기호열 R이 $P \Rightarrow Rd$이면서 $\ell(P)<\ell(R)$을 만족할 때 R은 P로부터 생성가능이라 하고 $P \rightarrow R$로 표기한다.

이 생성가능 조작을 반복함으로써 임의의 모든 기호열 $X \in A^*$는 λ로부터 생성할 수 있다. 예를들면, X=aababbb는

$\lambda \rightarrow a$
$a \rightarrow a \cdot ab$
$aab \rightarrow a \cdot ab \cdot ababb$

로 생성분해 할 수 있다. 일반적으로 길이 n의 기호열 $X_n \in A^*$는 생성관계에 의해 귀납적으로 $X_n \rightarrow H(X)=X_1 \cdots X_m$, $X_i=X(h_{i-1}+1, h_i)$, $m<n$, $1 \le i \le m$, $h_0 \equiv 0$, $h_1 \equiv 1$, $h_m=\ell(X)$로 분해되며, 이 H(X)를 X의 production history라 하고 X_i를 H(X)의 성분(components)이라 한다. 기호열 $X \in A^*$의 임의의 H(X) 중의 성분의 수를 $C_H(X)$라 하면 Lempel-Ziv의 계산론적 정보량은 생성의 회수를 고려하여 다음 정의와 같이 주어진다.

[정의 3] Lempel-Ziv의 계산론적 정보량 $C_{LZ}(X)$는 $C_{LZ}(X)=\min\{C_H(X)\}$.

이때 Lempel-Ziv 정보량의 성질은 다음과 같이 정의된다.

[정의 4] 길이 n의 임의의 기호열 $X_n \in A^*$에 대해

$$C_{LZ}(X) < \frac{n}{(1-\varepsilon_n)\log(n)} .$$

여기서, ε_n은 $\lim_{n \to \infty} \varepsilon_n = 0$을 만족한다. 이 상한은 계열을 생성분해 했을때 서로 틀린 성분기호열의 최대수를 나타내며, Kolmogorov의 복잡도(complexity)가 n이 충분히 커지면 거의 n과 같아짐에 비해 Lempel-Ziv의 복잡도는 n/log n에 접근하게 되는 특징이 있다.

3. Ziv-Lempel부호

Ziv와 Lempel에 의해 고안된 이 부호(이하 ZL부호)는 여러 분야에 응용이 활발한 가운데 특히, 텍스트 압축에 널리 이용되고 있다. 이론적으로는 부호기와 복호기의 메모리에 제한이 없다면 정상 에르고딕 정보원(stationary ergodic source)에 대해 점근적으로 최적(asymptotically optimum) 즉, 1기호의 평균부호의 길이가 엔트로피에 확률 1로 수렴함이 증명되어 있다[5]. 또한 수렴의 속도에 있어서 독립 정보원의 경우 $0(\log n/n)$[6], 마르코프 정보원에서는 $0(\log \log n/\log n)$임이 알려져 있다[7]. 그러나, 실제의 실현에 있어서는 유한 메모리로 한정시키지 않으면 안되기 때문에 이를 고려한 수 많은 변형이 제안되어 있다[8]. 여기서는 Ziv와 Lempel이 고안한 두개의 부호를 알고리즘을 중심으로 보이고 그 성능 개선을 위한 여러 변형들에 대해 알아본다.

ZL77방식[9]은 정보원 기호열을 부호화하면서 적응사전을 구축하여 일치하는 성분기호열에 대응하는 포인터를 부호화하는데 비하여 ZL78방식[5]은 증분분해(incremental parsing)를 이용하여 일치하는 부분열의 탐색범위를 제한함으로써 그 구성이 간단하고 고속화를 목적으로 하는 방식이다. 이들 부호의 점근적 최량성의 증명은 원문[5),9)]을 참고함이 좋을 것으로

사료된다.

가. ZL77방식

이 방식[9]은 입력기호열 $X=x_1x_2\cdots\in A^*$를 입력순으로 성분$\{X_i\}$로 분해(parsing)하는 과정과 분해된 $\{X_i\}$를 2진부호 $\{C_i\}$로 변환하는 부호화 과정으로 구분된다.

(1) 생성분해(production parsing)

$X\in A^*$의 proper prefix $X(1, j)$와 정정수(positive number) $1\le i\le j$에 대해 $X(i, i+\ell-1)=X(j+1, j+\ell)$, $\ell\le\ell(X)-j$를 만족하는 최대 정정수 ℓ을 $L(i)$로 하고 $L(p)=\max\{L(i)\mid 1\le i\le j\}$로 되는 $X(1, j)$내의 위치로 하여 X의 부분열 $X(j+1, j+L(p))$를 $X(1, j)$의 X에 재생확대로서 p를 부분열 $X(j+1, j+L(p))$의 포인터로 한다.

(2) 파라메타 설정(parameter setting)

파라메타 L_s를 정하여 $L_s\ge\ell(X_i)$, $i=1, 2, \cdots$,가 되도록 분해를 실시한다. X_i에 대응하는 C_i의 길이는 고정길이 L_c로 한다. 여기서는 N기호를 저장할 수 있는 버퍼B를 사용하여 최신의 입력 기호열을 보관한다. 따라서 부호어의 길이 L_c는

$$L_c=1+\lceil\log_\alpha(N-L_s)\rceil+\lceil\log_\alpha L_s\rceil,\ \alpha=|A|.$$

여기서, $\lceil x\rceil$는 x보다 큰 최소정수를 나타내는 ceiling함수이다.

(3) 초기화(initialization)

부호화의 초기시점에서 입력 기호열 X의 어두에 $N-L_s$개의 영의 열$Z=0^{N-L_s}$를 부가한다. 첫번째 성분 X_1을 결정하기 위하여 버퍼의 내용 B_1을

$$B_1=Z\cdot X(1, L_s)$$

로 한다.

(4) 내부호화(inner coding)

재귀적 절차에 따라 $i\ge1$번째의 버퍼 B_i를 설정한 후,

$$X_i=B_i(N-L_s+1, N-L_s+\ell(X_i))$$

의 어두 X_id가 $B_i(1, N-L_s)$의 $B_i(1, N-1)$로의 최대 재생확대가 되도록 X_i를 구한다.

(5) 외부호화(outter coding)

X_i를 결정하는 포인터를 P_i라 할때, X_i의 외부호어 C_i는,

$$C_i=C_{i1}\ C_{i2}\ C_{i3}$$

로써 C_{i1}은 $\lceil\log(N-L_s)\rceil$의 길이를 갖는 P_i-1의 α진 표현, C_{i2}는 $\lceil\log L_s\rceil$의 길이를 갖는 $\ell(X_i)-1$의 α의 표현, $\ell(X_i)=\ell_i$, C_{i3}은 X_i의 최우측의 기호, 즉 B_i의 $N-L_s+\ell(X_i)$번째 기호이다.

(6) 버퍼의 갱신(buffer update)

버퍼의 내용 B_i를 B_{i+1}로 갱신하려면 B_i의 최초의 $\ell(X_i)$개의 기호를 왼쪽으로 자리이동시키고 기호열 X의 다음의 $\ell_i=\ell(X_i)$개의 새로운 기호를 오른쪽에서 입력한다. 즉,

$$B_{i+1}=B_i(\ell_i+1, N)\cdot X(h_i+1, h_i+\ell_i)$$

로서 h_i는 B_i의 가장 오른쪽의 기호 $B_i(N, N)$이 B_{i+1}내에 위치하는 장소와 일치한다.

(4)~(6)의 단계를 입력 기호열이 끝날때까지 반복한다. 다음에 간단한 예를 나타낸다. $X_{24}=00101021021021021200$, $\alpha=3$, $L_s=9$, $N=18$일때, $L_c=1+\lceil\log_3(18-9)\rceil+\lceil\log_3 9\rceil=5$로 주어진다.

① 초기화(initialization)

$$B_1=\underbrace{000000000}_{\lceil N-L_s=9\rceil}\ \underbrace{001010210}_{\lceil L_s=9\rceil}$$

② 성분분해(parsing)

$X_1=001$, $p_1=9$, $\ell_1=3$

③ 부호어(codeword)

C_{11} : 22 C_{12} : 02 C_{13} : 1 → $C_1=22021$

④ 버퍼 갱신(buffer update)

$B_2=000000001\ 010210210\rightarrow X_2=0102$, $P_2=8$, $\ell_2=4$, $C_2=21102$

$B_3=000010102\ 102102120\rightarrow X_3=10210212$, $P_3=7$, $\ell_3=8$, $C_3=20212$

$B_4=210210212\ 021021200\rightarrow X_4=021021200$, $P_4=3$, $\ell_4=9$, $C_4=02220$

위 예제에서 알 수 있듯이 입력열의 초기에서는 부호계열의 길이가 입력계열의 길이보다 약간 짧아지나, 성분분해가 압축에 유효한 길이로 증가되면서 급속하게 줄어들므로 큰 압축효과를 기대할 수 있다.

(7) 복호화(decoding)

① 버퍼 설정

i번째 성분 X_i를 복호하기 위하여 길이 $N-L_s$의 버퍼 D를 설정하여, X_1을 위한 최초의 버퍼에는 $N-L_s$개의 영의 열 $Z=0^{N-L_s}$를 넣어둔다.

② 성분 X_i의 복호화

$1 \le i$번째 내부호어 C_i의 $\lceil \log_\alpha(N-L_s) \rceil$, $\lceil \log_\alpha L_s \rceil$개의 기호로부터 p_i, ℓ_i를 구한다.

③ D_i의 p_i위치에 있는 기호 $D_i(p_i, p_i)$를 끄집어내어 D_i를 왼쪽으로 1기호 이동하여 $N-L_s$ 위치에 복사한다. ℓ_i-1회 반복후에 C_{i3}를 최후미에 덧붙인다.

④ 버퍼 D_i의 갱신

버퍼의 내용은 D_{i+1}로 되고 $X_i=D_{i+1}(n-L_s-\ell_i+1, n-L_s)$로 된다.

②~④의 과정을 부호열 C가 없어질때까지 반복한다.

나. ZL78방식

이 방식[5]은 ZL77방식의 재생확대를 이미 정의된 내부호어 X_i의 선두에 한정시켜서 재생확대에 의한 산출조건을 완화시킨 기법이다.

(1) 증분분해(incremental parsing)

입력 기호열 X를 가변길이의 $p+1$개의 성분 $\{X_m\}$으로 분해하는 것으로 $X=X_1X_2 \cdots X_{p+1}$로 된다. 단, $X_m=X(n_{m-1}, n_m)$, $|X_m|=n_m-n_{m-1}$, $n_0=0$이다.

(2) 분해기준(parsing rule)

① X_1의 길이는 1이다.

② X_{p+1}를 제외한 모든 성분 $X_1, \cdots, X_p$는 서로 다르다.

③ 모든 m에 대해 X_m의 최후의 기호를 제외한 X_md와 일치하는 X_i, $i \in \{0, 1, \cdots, m-1\}$가 반드시 하나 존재한다.

(3) 성분 X_m의 부호화(encoding)

X_md와 일치하는 X_i와 X_m의 최후미 기호 $X_m(n_m, n_m)$로부터 C_m을 다음식과 같이 구한다.

$$C_m=i\alpha+X_m(n_m, n_m),\ i \le m-1,\ X_m(n_m, n_m) \le \alpha-1$$

이므로 C_m의 상한은

$$C_m \le (m-1)\alpha+(\alpha-1)=m\alpha-1$$

따라서, C_m을 2진 표현하기 위한 부호의 길이 L_m은

$$L_m=\lceil \log_2(m\alpha) \rceil,\ m=1, 2, \cdots$$

으로 된다. 이 2진계열은 부호계열 d_m으로 된다. 즉, 길이 n의 $X_n=X_1X_2 \cdots X_{p+1}$은 d_m를 연결한 계열 $D=d_1d_2 \cdots d_pd_{p+1}$으로 부호화 된다. 이때 D의 총 길이 L은

$$L=\sum_{m=1}^{p+1} L_m=\sum_{m=1}^{p+1} \lceil \log_2(m\alpha) \rceil.$$

(4) 복호화(decoding)

주어진 부호계열 D를 선두로부터 L_m 길이로 분해하면 $d_1, d_2, \cdots, d_{p+1}$이 얻어진다. 지금 $X_1, X_2, \cdots, X_m$까지 복호되었다면, d_{m+1}로부터 C_{m+1}이 구해져 X_{m+1}의 복원이 가능하게 된다.

다. ZL77방식의 변형(ZL77's variations)

(1) LZR(Lempel-Ziv-Rodeh)방식[10]

LZR방식은 현재 부호화하려는 기호열과 일치하는 부분열 X_i를 이미 부호화가 끝난 과거의 모든 부분열에 포인터를 허용하는 점 이외에는 ZL77방식과 동일하다. 〈i, j, a〉상의 위치포인터 i와 길이포인터 j의 값이 무한히 커질 수 있기 때문에 가변길이의 정수 표현에 의해 부호화[11]된다. $a \in A=\{0, 1, \cdots \alpha-1\}$는 성분 X_i의 최후미 기호이다.

LZR방식은 과거의 기호열을 등록하고 있는 적응사전의 크기에 제한이 없기 때문에 많은 기억공간을 필요로 하며 n기호 길이의 텍스트를 처리하는데 소요되는 시간이 $O(n)$이 걸리는 결점이 있다. 그 해결방안으로서 suffix tree[12]를 사용하여 선형 계산량으로 처리할 수 있는 알고리즘을 제시하고 있지만 같은 노력의 다른 방식에 비해 효과적이지 못한 평가를 받고 있다.

(2) LZSS(Lempel-Ziv-Storer-Szymanski)방식[13]

ZL77과 LZR방식의 출력은 치환 포인터와 최종기호를 나타내는 〈i, j, a〉의 세 파라메타로 구성된다. 그러나, 최종기호를 나타내는 $a \in A$는 다음 부분열 X_{i+1}의 일부분에 포함할 수 있기 때문에 그대로 전송함은 압축의 효과를 기대할 수 없게 된다. 따라서 LZSS방식에서는 포인터의 자유혼합방식(free mixture of pointers)을 사용하여 이를 해결하고 있다. 즉, N기호분의 윈도우를 사용하여 최대길이 일치부분열에 대한 포인터(offset, length)를 구하여 길이가 임계값 p보다 크면 포인터를 부호화하나 그렇지 않으면 부호화하려는 기호열의 최초의 기호를 그대로 출력한다. 이때, 포인터와 기호부분을 구별하기 위해 추가비트가 필요하므로 부호길이 L(m)은

$$L(m)=\begin{cases} 1+\lceil \log \alpha \rceil, & m \in A \\ 1+\lceil \log N \rceil+\lceil \log(F-p) \rceil, & \text{otherwise} \end{cases}$$

로 주어진다. 여기서 F는 부호화하려는 기호열의 최대길이로써 ZL77방식의 L_s에 대응하는 고정된 파라메타이다.

(3) LZB(Lempel-Ziv-Bell)[14]

LZSS방식의 모든 포인터는 부분열의 길이에 관계없이 고정길이의 L(m)으로 표현하고 있다. 그러나, 일치하는 부분열의 길이에 따라 발생빈도가 다르기 때문에 가변길이로 포인터를 표현함이 더욱 효과적이다. 이러한 문제점을 해결하기 위해 LZB방식에서는 위치를 나타내는 포인터 i에 대해 절약된 정규 2진법을 적용하고 있다. 또한 길이를 나타내는 포인터 j에 대해 정수 표현법(γ부호)[15]를 사용하여 일치하는 부분열의 길이 F에 제한을 두지 않는 방법을 취하고 있다. n기호의 부호화와 끝난 시점에서의 부호길이 L(m)은,

$$L(m)=\begin{cases} 1+\lceil \log \alpha \rceil, & m \in A \\ 1+\lceil \log \min(n, N) \rceil+2\lfloor \log |m| \rfloor+1, & \text{otherwise} \end{cases}$$

로 된다. 여기서 ⌊x⌋는 x보다 작은 최대정수를 나타내는 floor함수이다.

(4) LZH(Lempel-Ziv-Brent)[16]

LZB방식은 포인터를 2진 표현하기 위해 비교적 간단한 정수 표현법을 사용하고 있으나, 포인터를 효과적으로 표현하기 위해서는 확률분포를 이용함이 바람직하다. LZH방식은 제1단계에서 LZSS방식을 수행하고 제2단계에서는 LZSS방식에서 얻어진 포인터와 최종기호들의 발생확률을 이용하여 Huffman 부호화 한다. 즉, 적응사전은 LZSS방식과 같으나 부호길이 L(m)은 Huffman 알고리즘에 의해 결정된다. 이때 Huffman table은 N+F+α로 되어 수천에 이르는 크기이므로 LZH에서는 위치 포인터를 그룹화하여 부호어가 수백이 되도록 해결하고 있으나 ZL부호의 간결성과 고속성에 상반되는 결과를 낳는다.

(5) LZP(Lempel-Ziv-Park)[17]

ZL77방식은 생성가능(producible)의 분해법을 사용하고 있으나 LZP에서는 재생가능(producible)의 분해법을 이용하고 있다. 이것은 LZW부호가 성분분해는 생성가능 성질을 이용하나, 다음 부분열의 선두기호는 직전 부분열의 후미 기호와 중복시켜 압축이 되지 않은 상태로 처리됨을 방지하고 있는 것과 유사하다. 따라서, LZP방식은 ZL77방식의 VF(Variable-to-Fixed) 형식을 VV(Variable-to-Variable) 형식으로 개량한 형태가 된다. 즉, 압축에 유효한 길이(L_{th}) 이상의 성분분해 X_i가 이루어지면 그 시점 위치 i와 길이 j를 부호화하고, 그렇지 않으면 X_i의 선두 1기호만을 그대로 전송한다. 이때, 두 형태를 구분하기 위한 flag의 사용은 LZSS방식과 유사하다. 위치 $i(0 \le i < p)$와 길이 $j(L_{th} \le j \le q+L_{th})$는 각각 $\lceil \log_2 p \rceil$, $\lceil \log_2 q \rceil$ 비트로 표현되며, p는 적응사전의 크기, q는 X_i가 취할 수 있는 최대길이이다.

LZP방식의 또 다른 방법은 재생가능에 의한 LZ-VV방식에서 얻어지는 위치 i, 길이 j 및 비압축 기호 a에 대해 적응 산술부호화를 적응시키는 것이다. 그 이유는 LZB방식에서와 같이 〈i, j, a〉의 세 파라메타의 출현빈도의 편중에 의한 압축이 가능하기 때문이다. 이는 LZH방식이 Huffman 부호를 이용하는 점과 유

사하나, LZH방식은 이중 주사방식(2 pass)이나 LZP 방식은 단일 주사방식(1 pass)이며 압축률에서도 산술부호의 우수성이 입증되어 있기 때문에 LHZ 보다 효과적이라 본다.

라. ZL78방식의 변형(ZL78's Variations)

(1) LZW(Lempel-Ziv-Welch)방식[18]

ZL78방식이 LZW방식으로 변형되는 것은 ZL77방식이 LZSS방식으로 변형된 것과 유사하다. 즉, 각 부분열에 포함된 마지막 기호는 압축되지 않은 상태로 처리되기 때문에 압축효과를 기대할 수 없다.

LZW방식은 모든 알파벳 요소를 적응사전에 초기화시켜 두고 현재 부분열의 마지막 기호를 다음 부분열의 첫 기호에 중복시켜 처리하고 있다. 따라서, LZW방식은 적응 사전의 각 부분열을 일의적으로 식별할 수 있는 포인터만이 부호어로 된다. LZW방식은 고속 데이타 전송을 전제로 하는 디스크 채널상에서 하드웨어 실현을 목표로 제안되었기 때문에 압축률의 개선보다 고속으로 부호화하는 면에 촛점을 둔 방식이다. 따라서, 포인터는 12비트의 고정길이를 사용하고 있으며 적응사전이 포화가 되면 고정된 사전을 그대로 유지하면서 사용한다.

(2) LZC(Compress command)[19]

이 방식은 LZW방식의 실용화를 목적으로 수정된 것으로 그 성능의 우수함이 인정되어 UNIX시스템의 "compress" command로 채용되어 널리 사용되고 있다. 포인터가 나타낼 수 있는 최대길이는 사용 기억공간이 취할 수 있는 범위 이내로 제한하면서 포인터는 가변길이로 표현된다. 또한, 적응사전이 포화상태로 되면 압축비가 떨어지는 것을 감시하여 재초기화한다. 따라서 적응사전의 크기 M은,

$M=A\cup\{$지금까지 분해된 모든 부분열$\}$

로 되고 해당 부분열 m의 부호길이 L(m)은

$L(m)=\lceil \log |M| \rceil$

으로 된다.

(3) LZMW(Lempel-Ziv-Miller-Wegmen)[20]

ZL78로부터 유도된 다른 모든 방식에서는 기존의 부분열에 한 기호를 첨가하여 새로운 부분열을 작성하고 있다. LZMW 방식에서는 새로운 부분열을 바로 직전의 두개의 부분열을 연결하여 구성하고 있다. 따라서, 부분열의 어두가 사전내에 존재하지 않는 경우에도 비교적 빨리 긴 부분열을 구축할 수 있다는 잇점이 있다. 적응사전의 관리는 LRU(Least Recently Used) 방식을 채용하여 제한된 기억공간의 사용으로 정보원의 성질에 적응이 빨리 이루어질 수 있도록 하고 있다. 그러나 효과적인 연산을 위해 복잡한 데이타 구조를 사용해야 하는 결점이 있다.

(4) LZJ(Lempel-Ziv-Jakobsson)[21]

LZJ방식의 적응사전은 지금까지 처리된 계열내에 있는 모든 고유 부분열을 포함하게 되며 부분열의 길이는 $2\leq h\leq 6$로 제한된다. 이때 참조되는 부분열의 인덱스번호는 사전내의 모든 부분열을 일의로 식별할 수 있는 고정길이 부호를 사용하고 있다. LZJ방식에서 사용되는 적응사전 M은 $M=A\cup\{$처리가 끝난 계열에 있는 길이 2~h까지의 고유 기호열$\}$, $|M|\leq H$ 크기이다. 따라서 부호길이 $L(m)=\log_2 H$로 된다.

(5) LZFG(Lempel-Ziv-Fiala-Greene)[22]

LZFG방식은 기존의 변형된 ZL방식 가운데 가장 성능이 뛰어난 것으로 평가되고 있다. 이 방식은 PATRICIA TRIE구조[23]를 이용하여 포인터를 TRIE 상의 위치로 나타냄으로써 효과적인 표현과 고속처리를 수행하고 있다. LZFG방식에서는 참조기호열을 이미 분해가 이루어진 곳에서만 포인트할 수 있도록 제한하기 때문에 LZJ에 비해 상당히 빠른 특징이 있다.

ZL78방식과는 달리 포인터는 일치하는 부분의 길이를 나타내기 위하여 제한되지 않은 길이를 나타낼 수 있으며, 부호화가 끝난 기호들은 ZL77방식에서와 같이 같이 제한된 윈도우에 놓여진다. LZFG방식의 분해과정을 구체적인 예를 통하여 알아보자. $A=\{0, 1\}$, 윈도우크기 $|W|=4$의 경우, 입력기호열 X=01 01010110101을 분해하면 0, 1, 010101, 1, 0, 101의 부분열로 분해된다. 즉, $x_1=0$와 $x_2=1$은 참조하는 기호열이 없기 때문에 길이 1의 부분열로 분해된다. $x_3 \cdots x_8=010101$은 위치 $i=1$에서 시작되는 부분열 01을 3회 반복하면 생성된다. $t=4$에서 윈도우 $W=x_5 x_6 x_7 x_8=0101$에는 $x_9=1$과 일치하는 기호가 존재하지만 부분열의 선두가 아니기 때문에 x_9는 W로부터 참조할 수 없다. x_{11}의 위치에서 $W=x_7 \cdots x_{10}=0110$에는 x_9에서 시작하는 부분열이 존재하므로 $x_{11}x_{12}x_{13}$

=101으로 분해할 수 있다.

분해된 부분열을 가르키는 포인터는 참조가 이루어지는 가부에 의해 직접 모드와 간접모드로 부호화가 이루어진다. 즉, 직접모드에서는 flag=0에 이어 기호 자신을 $\lceil \log_2 \alpha \rceil$ 비트로 출력하고, 간접모드에서는 flag=1에 이어 참조 부분열의 위치 p와 그 길이 ℓ을 부호화한다. 구체적인 부호화는 PATRICIA 구조를 이용하여 효과적인 표현을 보이고 있다.

한편, LZFG방식에 의해 n이 k개의 부분열 W_1, W_2, …, W_k로 분해되어 부호화된 전체 부호길이 $\rho_M(X)$는,

$$\rho_M(X)=k+\sum_{j\in J_D}\lceil \log \alpha \rceil+\sum_{j\in J_I}\{\lceil \log m_j \rceil+2\lceil \log(\ell_j+1)\rceil\}$$

로 된다. 여기서, m_j는 W_j로 분해시 참조 가능한 선두 위치의 갯수이고, ℓ_j는 W_j의 길이이다. 또, J_D는 직접모드에 속하는 부분열의 번호, J_I는 간접모드에 속하는 단어의 번호로 $k=|J_D|+|J_I|$이다. 따라서, LZFG 부호는 유한 상태 정보원에 대해 O(log log M/log M)으로 점근적 최량임이 증명되어 있다[24]. 여기서 M은 길이 n의 입력 기호열 가운데 충분히 큰 값이다.

4. 결 론

Lempel과 Ziv는 Kolmogorov-Chaitin의 계산론적 정보량에 기초하여 기호의 복제를 기본조작으로 계산론적 정보량의 한 구성적 정의를 부여한 점에서 그 가치를 인정 받고 있다. 더우기, 이 발상을 이산정보의 무잡음 압축(noiseless compression)에 이용할 수 있는 부호화 방법을 고안하여 유니버샬 부호의 대명사로 불리어지게 되었으며 수 많은 후속 연구의 발판을 구축하게 되었다.

여기서는 Lempel-Ziv계산론적 정보량에 대하여 정의하고 그에 바탕을 두고 고안된 두가지의 ZL부호에 대해 알고리즘을 중심으로 소개하였다. 또한, ZL부호를 실용의 압축기법으로 응용하기 위한 여러가지 변형에 대해 알아 보았으며 앞으로의 이 분야의 연구에 조금이라도 도움이 되길 기대한다.

참 고 문 헌

1. S. Yamada, "Kolmogorov-Chaitin computational complexity and Lempel-Ziv universal algorithms for data compression", Univac Japan Technology Review, 1, pp.74-88(Aug. 1981) (In Japanese)

2. N. S. Jayant, P. Noll, "Digital coding of waveforms", Prentice-Hall, (1984)

3. D. A. Huffman, "A method for the construction of minimum-redundancy codes", Proc. IRE, 40, 9, pp.1098-1101(Sep. 1952)

4. A. Lempel and J. Ziv, "On the complexity of finite sequences", IEEE Trans. IT-22, 1, pp.7-81(Jan. 1976)

5. J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding", IEEE Trans. IT-24, 5, pp.530-536(Sep. 1978)

6. M. Yuri and J. Tjalkens, "The redundancy of the Ziv-Lempel algorithm for memoryless sources", 1990 IEEE Information Theory Workshop (June. 1990)

7. N. Merhav, M. Gutman and J. Ziv, "On the estimation of the order of a Markov chain and universal data compression", IEEE Trans. IT-35, 5, pp.1014-1019(Sep. 1989)

8. T. C. Bell, J. G. Cleary and I. H. Witten, "Text compression", Prentice Hall, Inc.,(1990)

9. J. Ziv and A. Lempel, "An universal algorithm for sequential data compression", IEEE Trans. IT-23, 3, pp.337-343(May. 1977)

10. M. Rodeh, V. R. Pratt and S. Even, "Linear algorithm for data compression via string matching", JACM 24, 5, pp.16-24(Jan. 1981)

11. S. Even and M. Rodeh, "Economical Encoding of commas between strings", CACM 21, pp.

315-317(1987)

12. E. M. McCreight, "A space-economical suffix tree construction algorithm", JACM 23, 2, pp. 262-272(Apr. 1976)

13. J. A. Storer and T. G. Szymanski, "Data compression via textual substitution", JACM 29, 4, pp. 928-951(Oct. 1982)

14. T. C. Bell, "Better OPM/L text compression", IEEE Trans COM-34, 12, pp. 1176-1182 (Dec. 1986)

15. P. Elias, "Universal code sets and representations of the integers", IEEE Trans. IT-21, 2, pp. 194-203(Mar. 1975)

16. R. P. Brent, "A linear algorithm for data compression", Australian Computer J., 19, 2, pp. 64-68(1987)

17. J. H. PARK et al, "Practical data compression methods using pattern matching and arithmetic codes", IEICE Trans. J71-A, 8, pp. 1615-1623 (1989)

18. T. A. Welch., "A technique for high-performance data compression", IEEE computer, 17, 6, pp. 8-19(June. 1984)

19. V. S. Miller and M. N. Wegman, "Variations on a theme by Ziv and Lempel", In Combinatorial algorithms on words, Springer-Verlag, N. Y., pp. 131-140(1985)

20. "Documentation of the COMPRESS command", In UNIX User's Manual, 4.2 BSD.

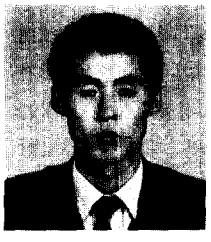
21. M. Jakobsson, "Compression of character strings by an adaptive dictionary", BIT, 25, 4, pp. 93-603(1985)

22. E. R. Fiala and D. H. Greene, "Data compression with finite windows", CACM, 32, 4, pp. 490-505(Apr. 1989)

23. D. Morrison, "PATRICA-practical algorithm to retrieve information coded in alphanumeric", JACM, 15, 4, pp. 514-534(Oct. 1968)

24. H. Morita and K. Kobayashi., "On asymptotic optimality of Fiala-Greeene codes", proc. of 14th SITA, pp. 489-492(Dec. 1991) (In Japanese)

□ 著者紹介



朴 志 煥(正會員)

1984年 2月 慶熙大學校 電子工學科(工學士)
1987年 3月 日本 國立電氣通信大學 大學院 情報工學科(工學修士)
1990年 3月 日本 横浜國立大學 大學院 電子情報工學科(工學博士)
1990年 3月～現在 釜山水產大學校 自然科學大學 電子計算學科 助教授