論 文

LZW 압축알고리즘의 개선에 관한 연구

正會員 丁 仙 伊* 正會員 鄭 鎭 旭*

A Study on the Improvement of LZW Compression Algorithm

Sun Ny JUNG*, Jin Wook CHUNG* *Regular Members*

要 約 현재 가장 널리 쓰이고 있는 압축알고리즘의 하나인 LZW 압축알고리즘은 이미 많은 연구에서 밝혀진 바 있듯이, 그 알고리즘 자체가 일련의 용장성을 내포하고 있다. 따라서 본 논문에서는 기존의 VF(Variable-to-Fixed)형 LZW 압축 알고리즘을 VV(Variable-to-Variable)형으로 개선함으로써, 그 용장성을 제거하였다. 결과적으로 본 개선알고리즘인 VV형이 VF형보다 압축효율, 특히 초기 압축효율에 있어서 더 우수함을 실험, 입증하였으며, 또 VV형의 압축후 출력비트열이 VF형보다 용장성이 더 작기 때문에 암호화에 이용시 보다 효과적임을 보였다.

ABSTRACT As proved in many studies, LZW algorithm, one of the widely used compression schemes, itself contains some types of redundancy, In this paper, VV-LZW compression algorithm is presented which encodes variable length source string into variable size codewords and reduces such redundancies that the original LZW scheme has. Experiment shows that this scheme especially has good initial compression efficiency and produces more complex output strings than the original LZW. This scheme is very useful to the data compression with small size, and the applications such as cryptography.

I. 서 론

최근 정보통신 기술의 발전에 따라 전자우편과 같은 많은 응용분야에서 취급되는 데이터의 양이 급격히 증가하고 있는 추세에 있다. 이와 더불어 컴퓨터 저장공간의 절약과 통신채널의 효율적 이용을 목적으로 한 데이터압축기술에 관한 중요성도 크게 부각되고 있다.

이미 데이터압축기술은 정보원 데이터의 확률구조에 기반한 통계적 압축기법을 비롯하여 확률구조와는 무관하게 모든 정보원 데이터를 처리할 수 있는 유니버셜 압축기법 등 많은 연구가 진행되어 오고 있다. 그 가운데 특히 유니버셜 압축기법의 대표적 기법으로서, Ziv-Lempel이 제안한 LZ알고리즘을 실용화한 LZW(Lempel-Ziv-Welch) 알고리즘[1]은 그 실용성이 널리 인정되어 CCITT의 전기통신표준인 V.42bis[2]로서 채택된 바 있으며, 그에 관한 개선연구[3][4]도 많이

*成均館大學校 情報工學科
Dept. of Infor. Eng., Sungkyunkwan Univ.
論文番號 : 90-70(接受 1990. 8. 7)

발표되어 있다.

그러나 최초로 Lempel-Ziv에 의해 제안[5]되어 지금까지 개발되어 온 대부분의 LZ계 알고리즘은 입력데이터가 충분히 길다는 가정하에서는 샤논이 제시한 데이터 압축한계에 점근적으로 근사한다는 점근적 최량성을 달성하고 있지만, 항상 고정된 최대 길이로 부호어를 부호화하기 때문에 초기 입력데이터에 대한 압축효과가 느리게 나타나며, 또 부호화 초기단계에서는 원래의 데이터보다 더 길게 되는 데이터확장이 발생하는 등의 문제점을 갖고 있다.

이러한 용장성을 제거하기 위한 연구로서는, LZ알고리즘에서 출력되는 부호어를 심볼로 간주하여 산술부호화하는 Rissanen의 유니버셜 알고리즘[6]이 있으며, LZ알고리즘의 증분분해를 이용하여 부호어의 2진 표현을 개선함으로써 LZ계의 용장성을 개선한 Hidetoshi Yokoo의 수정 Ziv-Lempel 부호[7]가 있다. 그러나 이 알고리즘들은 그 실용성에 있어서 압축수행시간과 데이터의 일반성에 있어서 해결해야 될 몇가지 과제가 남아 있다.

따라서 본 논문은 이미 그 실용성이 입증되어 있는 LZW알고리즘을 분석, 그 용장성에 대해 알아본 다음, 이를 제거하기 위해 Hidetoshi가 제안한 자연수의 2진 표현에 관한 기본 개념을 LZW알고리즘에 도입함으로써, 기존 VF(Variablet-to-Fixed) 형의 LZW 압축기법을 VV(Variable-to-Variable)형으로 개선, 구현하였다.

그 결과, 본 개선알고리즘이 기존 VF형 알고리즘보다 수행시간면에서 특별히 부가되는 오버헤드없이 더 우수한 압축효율을 보임을 실험을 통해 입증하였다. 특히 본 VV-LZW 알고리즘에 의한 압축열은 참고문헌[8]에서와 같이 암호알고리즘과의 결합시에 기존 VF형보다 키로서 더 안전성이 우수함을 보였다.

먼저 Ⅱ장에서는 VF-LZW알고리즘과 그 용장성에 대해서 설명하고, 다음으로 Ⅲ장에서는 VF-LZW알고리즘에 있어서 부호어의 2진 부호화과정시 접두어(prefix) 성질을 분석하여 부호어의 2진 부호화과정을 개선함으로써 VV-LZW알고리즘을 구현한다. 그리고 Ⅳ장에서는 VF-LZW와 VV-LZW알고리즘의 압축효율, 수행시간을 여러 실험데이터를 통해 비교하고, 양 알고리즘의 출력열이 갖는 complexity를 엔트로피와 용장성, 수열의 실험을 통해 비교한다. 마지막으로 Ⅴ장에서는 본 개선알고리즘이 압축효율의 향상뿐아니라, 암호코딩과의 결합시에도 효율적임을 확인하고, 그 발전방향을 제시한다.

Ⅱ. VF-LZW 압축기법과 용장성 분석

1978년 Ziv와 Lempel이 입력 비트열의 증분분해에 의한 압축알고리즘을 제안한 이후, 1984년 Welch가 이 알고리즘을 문자단위로 개선하여 실용화한 LZW(Lempel-Ziv-Welch)알고리즘은 현재 가장 널리 사용되고 있는 압축기법의 하나로서, 국제 전기통신자문위원회인 CCITT에 의해 V.42bis로 채택되어 있다.

본 장에서는 LZW압축알고리즘의 부호화 과정을 살펴보고, 부호화시의 용장성에 대해 분석한다. 이 알고리즘은 다음 Ⅱ장 1절에서 설명한 바와 같이 문자열 테이블내에 존재하는 가변길이의 입력문자열이 이미 할당된 부호어에 따라 고정길이로 2진부호화되기 때문에, 본 논문에서는 Ⅲ장에서의 개선알고리즘과 구분하기 위해 편의상 이를 VF(Variable-to-Fixed) LZW라 한다.

1. VF-LZW의 부호화과정

VF-LZW알고리즘은 최초의 LZ알고리즘과 마찬가지로 증분분해(incremental parsing)에 의해 입력데이터열을 분해하지만, LZ가 증분분해후 부호화하는 반면, LZW는 부호화후 증분분해하고 있다는 점이 다르며, 또 비트열이 아닌 문자열을 취급하고 있다. 이 알고리즘은 문자열 테이블(string table)을 유지하면서 문자열 테이블내에 존재하는 최대 길이의 입력문자열을 고정길이, 통상 12비트로 2진 부호화하는 것으로,

일반적으로 문자열 테이블내에 존재하는 모든 문자열은 테이블내의 다른 문자열과 서로 접두어 성질(prefix property)을 갖는다.

처음 제안된 VF-LZW알고리즘은 가변길이의 입력문자열을 모두 12비트의 고정길이로 2진 부호화하였으나, 그후 초기 압축효율의 개선을 위해 9비트부터 시작하여 문자열 테이블내에 할당된 부호어의 범위에 따라 12비트까지 2진 부호화하도록 개선된 VF-LZW알고리즘이 주로 사용되고 있으므로, 본 장에서는 이 후자의 알고리즘을 그 대상으로 하였다.

1.1 VF-LZW의 인코딩 과정

그 기본 동작메카니즘을 살펴보면, 먼저 입력되는 정보원을 문자(8비트)단위로 받아들여 증분분해에 의해 부분열(substring)을 생성함과 동시에 각 부분열에 부호어(code word)를 할당하여 두고, 그 부분열을 참조할 때마다 미리 규정된 코드길이로 이 부호어를 2진화하여 출력하는 것이다.

VF-LZW 압축알고리즘의 인코딩(encoding) 절차는 다음과 같다.

1) 문자열 테이블 초기화 단계:

8비트 단위의 모든 발생가능한 문자 256개의 단일 문자스트링을 문자열 테이블에 생성하고 그 10진값(ordinal number)인 0-255를 부호어로 할당한다. 그리고 다음에 생성될 문자열에 할당하기 위한 부호어변수인 C1의 값을 256으로 초기화한다.

2) 최초의 입력문자를 읽어 문자열(ω)로 취한다.

3) 다음 입력문자를 읽어 캐릭터(k)로 한다.

i) 다음 입력이 없으면 문자열 테이블로 부터 2)의 문자열에 할당된 부호어를 찾아, 이를 $\lceil \log_2(C1+1) \rceil$비트로 2진 부호화하여 출력하고, 수행을 멈춘다.

ii) ωk가 문자열 테이블에 있으면, ωk를 새로운 문자열(ω)로 하고 3)을 다시 수행한다.

iii) ωk가 문자열 테이블에 없으면, 문자열 ω에 할당된 부호어를 테이블에서 찾아 이를 $\lceil \log_2(C1+1) \rceil$비트로 2진 출력한 다음,

가) 문자열 테이블에 ωk에 대한 새로운 문자열을 생성(update)하여 부호어 C1을 할당하고, 다음 문자열의 부호어 할당을 위해 C1을 1증가시킨다.

나) 그 다음, 캐릭터 k를 다시 새로운 문자열로 취하고 3)으로 수행제어를 옮긴다.

(본 논문에서 표현하고 있는 $\lceil X \rceil$는 X이상의 최소정수값을 의미하며, $\lfloor X \rfloor$는 X이하의 최대정수값을 의미한다.)

위 알고리즘은 매번 $\lceil \log_2(C1+1) \rceil$비트로 부호화되므로 마치 가변길이로 부호화되는 것처럼 보이지만, C1의 값이 256부터 511까지는 9비트, 512부터 1023까지는 10비트, 1024부터 2047까지는 11비트, 그리고 2048부터 4095까지는 12비트로 부호어를 출력하도록 하여 각 범위내에서는 고정길이로 2진 부호화하고 있기 때문에 VF-LZW라 한다. 또 문자열 테이블에 생성되는 문자열의 갯수는 사용하는 시스템의 허용능력에 따라 확장할 수 있으나, 본 논문에서는 통상 많이 사용되고 있는 4096을 최대 문자열 테이블 엔트리로 제한하여 4096부터는 새로이 재초기화(reinitialize)하도록 함으로써 부호어에 대한 최대 비트수를 12비트로 제한하였다.

1.2 VF-LZW의 복호화 과정

복호화과정은 부호화와 거의 동일하나, 리커시브 디코딩(recursive decoding)을 해결키 위해 stack을 이용하였다. 또 부호화 과정시 바로 직전에 만들어진 부호어를 출력했을 경우에는 복호화시에 있어서 아직 정의되지 않은 부호어가 출현되므로, 이러한 특별한 경우를 처리하도록 하였다.

디코딩 절차는 다음과 같다.

1) 디코딩 초기화 단계:

인코딩시와 마찬가지로 8비트 단위의 모든 발생가능한 문자 256개의 단일 문자스트링을 문자열 테이블에 생성하고 그 10진값(ordinal number)인 0-255를 부호어로 할당한다. 그리고 C1의 값을 256으로 초기화한다.

2) 최초 입력비트열의 MSB로 부터 $\lceil \log_2 (C1+1) \rceil$비트를 읽어 이를 10진값으로 바꾸고, 이를 CODE와 OLDcode로 할당한다. 문자열 테이블 가운데 이 CODE를 부호어로 갖는 캐릭터 k를 출력하고, 이 k를 FINchar로 취한다.

3) 입력비트열로 부터 다시 $\lceil \log_2 (C1+1) \rceil$비트를 읽어 다음 부호어를 취하여, 이를 CODE와 INcode로 할당한다.

i) 만일 입력비트열로 부터 $\lceil \log_2 (C1+1) \rceil$비트를 취할 수 없으면, 즉 end-of-file이면 수행을 중지한다.

ii) 특별한 경우에 대한 처리:

만일 CODE가 문자열 테이블내에 정의되어 있지 않다면, FINchar를 출력하고, OLDcode를 CODE로 할당한다. 또 나중에 스택의 pop작업시 정상인 경우와 구분하여 처리하기 위해 플래그를 1로 세트하여 둔다.

4) CODE값의 체크:

i) CODE가 255보다 크다면, 테이블로 부터 이 CODE값을 부호어로 갖는 엔트리를 찾아서 그 엔트리의 k를 스택에 push하고, 다시 그 엔트리의 접두어 문자열(prefix code) ω를 CODE로 할당하여 단계 4)를 반복수행한다.

ii) CODE값이 255보다 작거나 같으면, 다시 플래그값을 체크하여 플래그가 0이면 아래 단계 가)와 나)를 차례로 수행하고, 플래그값이 1이면 아래단계 나)를 먼저 수행한 다음에 가)를 수행하고 단계 5)로 넘어간다.

가) 문자열 테이블에서 CODE값을 부호어로 갖는 단일 문자 엔트리의 k를 출력하고, k를 FINchar로 할당한다.

나) 위 단계 4)의 i)에서 스택에 push해 놓은 각 문자를 pop한다.

5) 테이블 엔트리의 생성: OLDcode와 k를 새로운 문자열로 생성하여 문자열 테이블내에 새로운 엔트리를 만들고 이 엔트리에 C1를 부호어로 할당한 다음, C1의 값을 +1한다.

i) 만일 C1의 값이 4096이면, 재초기화하기 위해 단계 1)로 수행제어를 옮긴다.

ii) C1의 값이 4096보다 작으면, INcode를 OLDcode로 하고 위 단계 3)으로 수행제어를 옮긴다.

2. VF-LZW 부호의 용장성

앞의 Ⅱ장 1절에서 살펴본 바와 같이 VF-LZW는 원래의 LZ부호와 마찬가지로 초기 입력데이터부분에 대한 압축효과가 다른 알고리즘에 비해 대단히 늦게 나타나며, 부분적으로는 데이터의 압축이 아닌 확장도 발생하게 되어, 입력데이터의 길이가 충분히 긴 경우에만 점근적 최량성을 갖는다는 사실을 알 수 있다.

먼저, 부호화 과정에서 보듯이 VF-LZW알고리즘은 문자열 테이블내에서 단일문자를 문자열로 하는 부호어, 즉 부호어가 0부터 255까지 문자열을 참조하는 경우에도 이를 $\lceil \log_2 (C1+1) \rceil$비트로 2진 출력하기 때문에 8비트의 원데이터길이를 최소 9비트에서 부터 12비트까지 확장 부호화하게 되는 모순이 발생하게 된다. 따라서 입력데이터의 초기부분에서는 데이터의 확장이 발생하고 있다.

또한 부호화시에 C1값이 255부터 4096까지 진행되는 동안 발생하는 모든 문자열의 부호어는 $\lceil \log_2 (C1+1) \rceil$비트로 2진 표현되고 있다. 즉 C1의 값이 256부터 511까지에서는 모든 문자열의 부호어는 9비트로 표현되며, 512부터 1023은 10비트, 1024부터 2047은 11비트, 2048부터 4095까지는 12비트로 표현된다. 그러나 여기서 표현되는 압축부호길이는 각 범위내에서 (C1+1)이 2의 정수승일때만이 실제 해당되는 길이로서, 나머지 부호어에 대해서는

$$L_{eff}(C1) = \log_2 (C1+1) \qquad (1)$$

의 비트가 실제 필요하게 됨을 알 수 있다. 식 1)에서 $L_{eff}(C1)$은 각 부호어 C1에 할당되어야 할 실효 부호길이를 나타낸다.

이상에서 살펴본 바와 같이 VF-LZW부호는 2가지 관점에서 부호화 자체에 상당한 용장성을 내포하고 있음을 확인할 수 있다.

Ⅲ. VF-LZW의 용장성 제거에 의한 VV-LZW 압축기법

본 논문에서는 2장에서 확인된 VF-LZW압축기법의 용장성 제거를 위해, VF-LZW에 있어서 각 C1의 값에 따라 발생가능한 모든 문자열, 즉 C1개(0- (C1-1))의 문자열에 대한 각 비트열을 조사하여, 그 비트열이 갖는 성질을 이용, VF-LZW를 가변길이 압축부호화할 수 있도록 한계값을 정하였다. 이에 따라 C1의 각 범위내에서 발생되는 모든 문자열에 대한 압축부호길이를 VF-LZW 부호화에서 처럼 $\lceil \log_2(C1+1) \rceil$비트로 일정하게 고정하지 않고, 매 문자열의 부호어값이 본 논문에서 정한 한계값 범위내에 속하는지의 여부에 따라 $\lceil \log_2 C1 \rceil$비트와 $\lceil \log_2(C1+1) \rceil$비트로 각각 가변길이 부호화하도록 하였다.

먼저, 본장에서는 한계값의 설정과정을 설명하고, 다음으로 본 논문에서 C언어를 이용하여 386시스템상에서 구현한 가변길이 부호화 알고리즘의 부호화 과정을 설명한다. 이 알고리즘을 본 논문에서는 VV(Variable-to-Variable)-LZW 압축기법이라 한다.

1. 문자열의 부호어에 대한 2진 표현방법

여기에서 도입하고 있는 기본적인 개념은 Hidetoshi Yokoo의 수정 Ziv-Lempel부호에서 참조한 것으로, 본 논문에서는 이를 확장, 구현하였다.

VF-LZW부호에서 살펴보았듯이, 다음에 생성될 문자열에 할당할 부호어 변수가 C1일 때 발생가능한 문자열의 부호어 범위는 0부터 C1-1까지 C1개의 정수값이다. 본 논문에서는 앞절에서 확인한 용장성을 제거하기 위해서 발생가능 부호어 0부터 C1-1까지의 값중에서 어떤 한계값 X를 정하여 0부터 X까지는 $\lceil \log_2 C1 \rceil$비트로 2진화하고, (X+1)부터 (C1-1)까지는 $\lceil \log_2(C1+1) \rceil$비트로 2진화하도록 하였다. 그런데 여기서 $\lceil \log_2 C1 \rceil$비트나 $\lceil \log_2(C1+1) \rceil$비트로 부호화된 부호어를 디코딩시에 올바로 얻어내기 위해서, 즉 순시복호를 하기 위해서는 (X+1)부터 (C1-1)까지의 부호어를 그대로 $\lceil \log_2(C1+1) \rceil$비트로 2진 부호화해서는 안된다. 따라서 0부터 X까지의 부호어에 대한 2진 표현과 (X+1)부터 (C1-1)까지의 2진 표현을 구분하기 위해, (X+1)부터 (C1-1)까지의 부호어에 (X+1)을 각각 더하여 (2X+2)부터 (C1+X)의 정수로 변환한 다음, 이 값을 $\lceil \log_2(C1+1) \rceil$비트로 2진 표현해야 한다. 한편 여기서 주의해야 할 점은 2진 표현해야 할 정수중에서 최대값인 (C1+X)가 $\lceil \log_2(C1+1) \rceil$비트로 표현할 수 있는 최대 정수인 $(2^{\lceil \log_2(C1+1) \rceil}-1)$보다 커서는 안되므로,

$$(C1+X) \leqq 2^{\lceil \log_2(C1+1) \rceil}-1$$

을 만족하는 X값중에서 어떤 값을 한계값으로 취해야 한다. 위 식을 다시 X에 대해서 정리하면, 다음 식2)와 같이 표현할 수 있다.

$$X \leqq 2^{\lceil \log_2(C1+1) \rceil}-C1-1 \qquad (2)$$

그런데 압축효율을 좋게 하기 위해서는 위 식2)를 만족하는 X값중에서 가장 큰 값을 정해야만 하므로, 실제 본 논문에서 규정한 한계값 X(이하, X를 한계값 변수 limit로 표현한다)를 다음 식 3)과 같이 정하였다.

$$\text{limit}=C3-C1-1 \quad (\text{여기서 } C3\text{는 } 2^{\lceil \log_2(C1+1) \rceil} \text{이다.}) \qquad (3)$$

따라서 본 논문에서는 인코딩시에 매 C1의 값에 있어서 발생되는 문자열의 부호어를 조사하여 이 한계값보다 작거나 같으면, 부호어를 $\lceil \log_2 C1 \rceil$비트로 출력하고, 한계값보다 크면 복호화시 순시복호를 위해 부호어에 (limit+1)의 값을 더하여 이를 $\lceil \log_2(C1+1) \rceil$비트로 출력하도록 하였다.

한편 복호화시에는 일단 입력되는 비트열중에서 먼저 MSB로부터 $\lceil \log_2 C1 \rceil$비트만을 취하여 그 10진값이 위 한계값 범위내에 들면 그 값이

바로 압축된 부호어가 되며, 한계값보다 크거나 같으면 다시 입력비트열중에서 1비트를 더취해 모두 $\lceil \log_2(C1+1) \rceil$비트의 10진값에서 limit를 뺀 값을 부호어로 한다.

일예로 C1이 520일 경우에 있어서, 본 알고리즘의 부호어 계산을 살펴본다.

예) C1=520일 때,

발생가능한 문자열의 부호어 범위: 0부터 519까지

limit=1024-520-1=503

[부호화] 만일 참조된 문자열의 부호어가
0부터 503까지 일 때는 9비트로 부호화하고
504부터 519까지는 각 부호어에 504를 더하여 10비트로 부호화한다.

[복호화] 입력된 비트열로 부터 9비트를 읽어서 그 값이
504보다 작으면, 그 값이 바로 부호어가 되며,
크거나 같으면, 입력비트열에서 다시 MSB부터 10비트를 읽어서 이 값에서 504를 뺀 값을 부호어로 한다.

이 예에서의 부호어 2진 표현을 비트열로 나타내면 그림 1과 같다. 그림에서 보듯이, 이 가변길이 부호화는 순시복호가 가능하도록 한계값을 전후한 비트표현이 서로 접두어성질(prefix property) 을 갖도록 되어 있음을 알 수 있다. 따라서 복호화시에는 단지 MSB로 부터 $\lceil \log_2 C1 \rceil$비트만을 취하여 비교해 봄으로써, 압축된 가변길이를 알 수 있게 된다.

2. VV-LZW 압축기법의 구현

본 논문에서는 앞의 Ⅲ장 1절에서 설명한 부호어의 2진 표현을 VF-LZW에 채용함으로써, 이 압축기법을 C언어를 이용하여 VV-LZW로 수정, 구현하였다.

먼저, 부호화 절차에서는 Ⅱ장의 1.1절과 다른 과정은 모두 동일하나, 단지 3)단계의 i)과 iii)부분 대신에, 문자열에 할당된 부호어를 찾아 2진화하는 부분에 다음 과정을 추가하기만 하면 된다.

i) 다음 입력이 없으면 문자열 테이블로 부터 2)의 문자열에 할당된 부호어를 찾아, 이 부호어가 limit값보다 작거나 같으면 이를 $\lceil \log_2 C1 \rceil$비트로 2진 부호화하며, limit값보다 크면 부호어에 limit+1을 더하여 이를 $\lceil \log_2(C1+1) \rceil$비트로 2진 부호화하여 출력하고, 수행을 멈춘다.

ii) ωk가 문자열 테이블에 없으면 문자열

발생가능 문자열 부호어	2진 표현	2진표현후 10진값
0	0 0 0 0 0 0 0 0 0 0	0
1	0 0 0 0 0 0 0 0 0 1	1
2	0 0 0 0 0 0 0 0 1 0	2
·	·	·
·	·	·
503	1 1 1 1 1 0 1 1 1	503
504	1 1 1 1 1 1 0 0 0 0	1008(504+504)
505	1 1 1 1 1 1 0 0 0 1	1009(504+505)
·	·	·
·	·	·
519	1 1 1 1 1 1 1 1 1 1	1023(504+519)

이 값들은 503보다 큰 것임.

이 최대값이 1023을 넘어서는 안됨.

그림 1. 문자열 부호어의 2진 표현예 (C1=520일 경우)
Fig. 1. Binary Representation of the String Codewords

ω에 할당된 부호어를 테이블에서 찾아 이 부호어가 limit값보다 작거나 같으면 이를 $\lceil \log_2 C1 \rceil$비트로 2진부호화하며, limit값보다 크면 부호어에 limit+1을 더하여 이를 $\lceil \log_2 (C1+1) \rceil$비트로 2진 부호화하여 출력한 다음,

위 과정을 인코드하고 있는 부분을 그림 2에 나타내었다.

```
transfer (unsigned long c1)
{
  limit=c3_table_count_1;
  if (c1<=limit) {
     shift=32-(c2-1)-bit_count;
     buffer +=(c1<<shift);
     bit_count +=(c2-1);
  }
  else {
     c1+=(limit+1);
     shift=32-c2_bit count;
     buffer +=(c1<<shift);
     bit_count +=c2;
  }
  while (bit_count>7) {
     temp=(buffer)>>24);
     putc((unsigned char)temp, output_file);
     output_count++;
     bit_count -=8;
     buffer <<=8;
```

그림 2. VV-LZW 압축기법의 가변길이 인코드화 과정
Fig. 2. VV-LZW Encoding Process

한편 복호화 과정도 Ⅱ장의 1.2절과 동일하며, 단지 입력비트열로 부터 부호어를 취하는 과정, 즉 단계2)와 3)에서 부호어를 취하는 부분만이 Ⅲ장의 1절에서 언급한 바와 같이 구현되어 있다. 이 과정을 C언어로 표현한 것을 그림 3에 나타내었다.

Ⅳ. VV-LZW 압축기법의 성능비교

가변길이로 압축부호를 생성하는 VV-LZW 압축기법이 갖는 성능개선 정도를 관찰하기 위해

```
unsigned get_code()
{
  unsigned c1=0, limit=0;
  unsigned long temp;
  unsigend long fetch;
  while (bit_count <c2) {
     temp=getc(input_file);
     shift=24-bit_count;
     buffer+=(temp << shift);
     bit_count +=8;
  }
  fetch=((unsigned long) (c3/2-1)) << (33-c2);
  c1=(buffer & fetch) >> (33-c2);
  limit=c3-code_count-1;
  if (c1>=limit) {
     fetch=((unsigned long) (c3-1)) << (32-c2);
     c1=(buffer & fetch) >> (32-c2);
     c1-=limit;
     buffer <<=c2;
     bit_count -=c2;
  }
  else {
     buffer <<=(c2-1);
     bit_count -=(c2-1);
  }
  return (c1);
```

그림 3. VV-LZW 압축기법의 가변길이 디코드화 과정
Fig. 3. VV-LZW Decoding Process

본 논문에서는 먼저, VF-LZW와 VV-LZW 각각의 압축문자열에 대한 평균부호장을 구하여 서로 비교하고, 다음으로 실제 몇가지 부류의 샘플화일에 대해 두 압축기법에 대해 초기 입력데이터의 압축효과와 총 압축비, 그리고 VV-LZW의 수행시간에 대한 오버헤드를 실험하였다. 사용된 샘플화일은 한글과 영문이 혼합된 텍스트화일, C언어로 쓰여진 프로그램 원시화일, 수행화일 등을 그 대상으로 하였으며, 실험을 위해 386시스템을 사용하였다.

또한 VV-LZW 압축기법에 의해 출력되는 압축된 비트열이 VF-LZW와는 달리 가변길이를 갖는 특성때문에, LZW계열의 압축알고리즘을 암호코딩과 결합하는 경우보다 우수한 안전성을 갖게 됨을 가변길이 비트열에 대한 엔트로피, 용장성등의 관점에서 입증하였다.

먼저, VF-LZW와 VV-LZW의 각 문자열에

대한 압축후 평균부호장을 다음 식 4)에 의해 구하고 식 5)에 의해 두 압축기법의 압축성능비교를 행하였다.

$$L=(\sum l_i)/n \quad (4)$$

$$\eta=L_{VV}/L_{VF} \quad (5)$$

(여기서 L은 문자열에 대한 평균부호장, $\sum l_i$는 각 문자열 부호길이의 총합, n은 압축후 출력되는 총 문자열의 갯수이며, η는 VV-LZW의 성능개선비로서 VV-LZW의 평균부호장인 L_{VV}와 VF-LZW의 평균부호장인 L_{VF}의 비로서 나타낸 것이다.)

위 실험결과, 각 압축기법에 대한 평균부호장은 표 1에서 보는 바와 같이 각각 11.10과 10.67이며, 성능개선비는 0.96로서, VV-LZW에 의해 압축하는 경우가 문자열당 평균부호장이 보다 짧음을 확인하였다. 그리고 생성 문자열의 갯수증가에 따른 성능개선비의 변화는 그림 4에 나타내었듯이, 초기에 약 수10개의 문자열이 생성되는 동안에는 그 성능이 급격히 향상되며, 그 이후에는 점차적으로 1에 접근하였다.

각 샘플화일에 있어서 VF-LZW와 VV-LZW에 대한 평균 압축효율은 표 2에 나타낸 바와 같이 VV-LZW측이 전체적으로는 약 4%정도 우수하지만, 그림 5의 입력 데이터 크기순으로 본 압축효과의 변화에서 보듯이 VV-LZW는 초기 입력데이터에 대한 압축효과가 VF-LZW에 비해 상당히 개선되었음을 알 수 있다. 따라서 본 논문에서 구현한 VV-LZW 압축기법이 원래의 VF-LZW알고리즘이 갖는 초기 입력데이터에 대한 압축효과의 지연을 개선하고 있을 뿐만아니라, 앞에서 언급한 단일문자에 대한 데이터 확장문제도 동시에 해결하고 있음을 알 수 있었다.

표 1. VF-LZW와 VV-LZW에 있어서 문자열에 대한 평균부호장*1과 성능개선비

Table 1. Average Codeword Length for a string and Performance Improvement Ratio of the VV-LZW relative to the VF-LZW

구 분	VF-LZW	VV-LZW	성능개선비
텍스트화일	11.23	10.87	0.97
C 소스화일	10.90	10.55	0.97
수행화일	11.17	10.58	0.95
평 균	11.10	10.67	0.96

주) *1: 평균부호장의 단위는 (bits/부호어)이다.

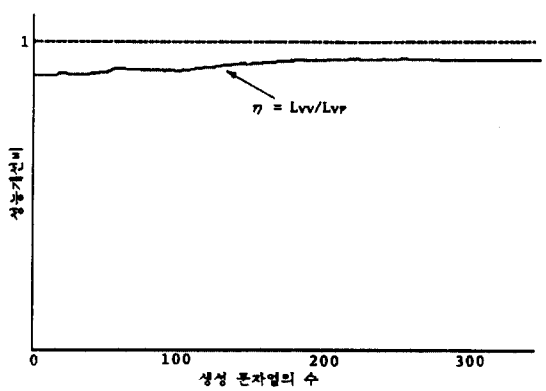

그림 4. VV-LZW 압축기법에 있어서 생성 문자열 증가에 따른 성능개선비 변화

Fig 4. The Effect of Output String Size on the Performance Improvement ratio in the VV-LZW

표 2. VF-LZW와 VV-LZW의 평균 압축효율*1

Table 2. Average Compression Ratios of the VF-LZW and the VV-LZW

구 분	VF-LZW	VV-LZW
텍스트화일	2.99	3.10
C 소스화일	2.57	2.65
수행화일	1.25	1.32

주) *1: 압축효율은 (압축전 데이터크기/압축후 데이터크기)이다.

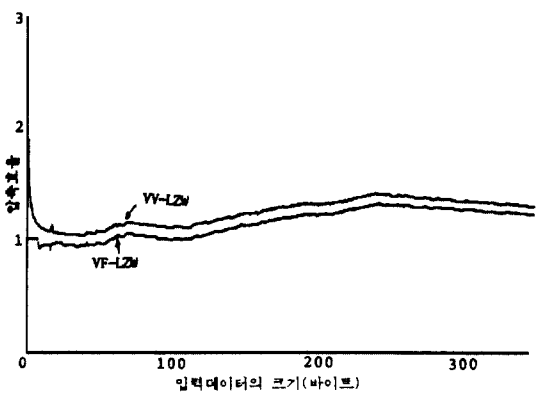

그림 5. 입력데이터의 크기에 따른 압축효율의 변화

Fig. 5. The Effect of Data Size on Compression Efficiency

또한 수행시간에 있어서 오버헤드는 표 3에 보였듯이 인코딩시간은 VF-LZW나 VV-LZW가 같으나 디코딩시간에 있어서는 VV-LZW측이 미소한 오버헤드를 나타내었다. 또 압축알고리즘의 수행시 차지하는 메모리비율은 두 압축기법이 모두 약 25k바이트로서 같게 관찰되었다.

표 3. VF-LZW와 VV-LZW 압축기법의 평균 수행시간[*1]
Table 3. Average Processing time of the VF-LZW and VV-LZW

구 분	VF-LZW	VV-LZW
텍스트화일	0.15 / 0.25	0.15 / 0.27
C 소스화일	0.16 / 0.25	0.16 / 0.26
수행화일	0.23 / 0.16	0.23 / 0.18

주) *1: 수행시간은 (인코딩시간 / 디코딩시간)을 나타내며, 단위는 ms이다.

한편 LZW계 압축알고리즘과 암호코딩을 결합하는 경우에 있어서는 기존의 VF-LZW계보다는 가변길이로 압축비트열을 출력하는 VV-LZW 알고리즘이 보다 효과적임을 알 수 있었다. VV-LZW에 있어서는 압축후 출력되는 비트열을 복호화할 때에 반드시 먼저 비트열의 MSB로 부터 $\lceil \log_2 C1 \rceil$비트를 읽어 그 값이 한계값에 속하는지의 여부를 확인한 이후에야 그 부호어가 $\lceil \log_2 C1 \rceil$비트 또는 $\lceil \log_2 (C1+1) \rceil$비트중 어떤 길이로 압축되었는지를 알 수 있게 된다. 그러나 암호화에 의해 $\lceil \log_2 C1 \rceil$비트의 값이 한계값에 대한 정당성을 부여할 수 없도록 임의로 변하게 되어, 암호해독자는 암호열로 부터 각 심볼의 구분을 찾을 수 없게 된다. 따라서 이미 알려진 바와 같이 LZW계 압축알고리즘은 에러에 대한 취약성이 절대적이므로(catastrophic failure), 암호공격자가 정확한 부호어의 길이를 찾아내기란 거의 불가능하다. 결과적으로 각 심볼이 고정길이로 압축되는 알고리즘보다는 가변길이로 압축되는 Fixed-to-Variable이나 Variable-to-Variable 압축알고리즘이 암호화시에는 더 효과적임을 알 수 있다.

암호화 관점에서 VV-LZW의 압축후 출력비트열을 참고문헌(8)에서와 같이 암호키로서 사용한다고 하였을 때, 이 출력비트열이 암호키로서 가치를 갖는지를 실험하기 위해, 본 논문에서는 VF-LZW와 VV-LZW의 출력비트열을 심볼당 8비트 단위로 간주하여 두 압축기법에 대한 엔트로피와 용장성 등을 다음 식4)- 식 7)에 의해 구하였다. 이 실험은 키열이 균등하게 분포될수록 암호계가 더 안전하다[9]는 샤논의 이론에 의한 것이다.

(엔트로피)

$$H(K) = -\sum P(k_i) \log P(k_i) \quad (6)$$

(최대 엔트로피)

$$H_0(K) = \log k \quad (7)$$

(용장성)

$$D = H_0(K) - H(K) \quad (8)$$

(Unicity Distance)

$$U.D = H(K) / D \quad (9)$$

(여기서 k는 심볼의 갯수이다.)

샤논의 이론에 의하면 최대 엔트로피는 모든 심볼의 발생확률이 같을 때의 엔트로피로서, 최대 엔트로피와 각 열의 엔트로피차가 작을수록 용장도가 작으며, 이 열들은 균등(uniformity) 분포를 갖게 된다. 따라서 암호키열의 용장도가 0에 접근할수록, 그 암호계의 유일한 해를 찾기 위해 필요한 암호문의 양에 대한 척도인 Unicity Distance는 무한대가 되어 암호강도가 더욱 커지게 된다.

계산결과, 입력데이터가 유한한 경우에 있어서 VF-LZW와 VV-LZW 각각의 엔트로피는 7.82와 7.84이며, 최대 엔트로피는 두 압축기법 모두 8로서, VF-LZW보다는 VV-LZW측의 용장도가 보다 작으므로, 앞에서 언급했듯이 본 연구에서의 개서 알고리즘이 암호화측면에서 더 가치가 있음을 알 수 있다.

이상의 성능실험결과, 가변길이로 부호어를 2진 표현하는 VV-LZW 압축기법이 압축효율면

에서는 초기 입력데이터에 대한 압축효과가 VF-LZW에 비해 우수하며, 이에 따른 처리시간과 메모리면에서의 부가적인 오버헤드 또한 없음을 알 수 있었다. 그리고 압축후 출력비트열을 암호코딩과 결합하는 경우에 있어서는 VF에 의한 압축기법보다는 VV압축기법이 보다 우수한 안전성을 갖음을 확인하였다.

V. 결 론

본 논문에서는 기존의 VF-LZW가 갖는 용장성을 분석하여 그 용장성을 출력비트열의 접두어 성질을 이용함으로써, 압축할 부호어를 가변길이로 부호화할 수 있는 VV-LZW 압축기법을 구현하였다.

그 결과, 기존 LZW계 압축기법들이 갖는 초기 압축효과의 지연 및 확장을 개선할 수 있었으며, 특히 압축코딩과 암호코딩과의 결합시에는 FF(Fixed-to-Fixed), FV(Fixed-to-Variable), VF(Variable-to-Fixed)방식에 의한 압축기법보다는 본 개선알고리즘과 같이 VV(Variable-to-Variable) 방식이 더 우수한 암호특성을 갖음도 확인할 수 있었다. 따라서 본 VV-LZW 압축기법을 암호코딩과의 결합시에 응용한다면 더 뛰어난 암호강도를 얻을 수 있을 것으로 기대된다.

앞으로, 본 논문에서 적용한 1단계 한계값 체크보다는 2,3단계로 이를 확장할 수 있도록 보다 보완하거나, 모듈러 연산 등을 통하여 압축부호어의 10진값을 더 작은 수로 함으로써 압축효율을 보다 개선하는 방안도 고려할 수 있을 것이다.

참 고 문 헌

1. Welch T.A., "A Technique for High Performance Data Compression", IEEE Computer 17, 6, pp.8-19, 1984.
2. CCITT COM XVII-R 1-E, Proposed Draft Recommendation V.42bis, 15/8/89.
3. T.C. Bell, "Better OPM/L Data Compression", IEEE Trans Comm., Vol.COM-34, No.12, pp.1179-1182, 1986.
4. E.R. Fiala and D.H . Greene, "Data Compression with Finite Windows", Comm. of ACM, Vol.32, No.4, pp. 490-505, April 1989.
5. Ziv. J. and Lempel. A., "Compression of Individual Sequences via Variable Rate Coding", IEEE Trans IT-24, No.5, pp.530-536, 1978.
6. J. Rissanen, "A Universal Data Compression System", IEEE Trans IT-29, No.5, pp.656-664, 1983.
7. Hidetoshi Yokoo, "An Improved ZIv-Lempel Coding Scheme for Universial Source Coding", 일본전자통신학회 논문지, Vol.J68-A, No.7, pp.664-671. July 1985.
8. 정진욱, 우치수, "리얼타임 통신환경에 있어서 압축코딩과 암호코딩의 결합", 한국정보과학회 논문지, Vol.17, No.4, 7월 1990(게재예정).
9. C.F. Shannon, "Communication Theory of Secrecy Systems", Bell Systems Technical Journal, vol.28, pp. 656-715, Oct 1949.

丁 仙 伊 (Sun Ny JUNG) 正會員
1958年 5月13日生
1981年 : 韓國航空大學 通信工學科 卒業
1989~現在 : 成均館大學校 大學院 情報工學科 碩士과정
1981~1988年 : 月刊 電子科學 編輯長

鄭 鎭 旭 (Jin Uk CHUNG) 正會員
1946年 6月20日生
1974年 : 成均館大學校 電氣工學科 卒業 (工學士)
1979年 : 成均館大學校 大學院 電子工學科 卒業 (工學碩士)
1986年 : 서울大學校 大學院 計算統計學科 博士과정 수료 (計算學 전공)
1974~1981年 : 韓國科學技術研究所 研究員
1982~1985年 : 韓國科學技術院 시스템공학센터 데이터통신실장
1981~1982年 : Racal-Milgo Co. 研究院 (미국 플로리다 소재)
1985~現在 : 成均館大學校 情報工學科 副教授