전략 테이블과 유전 알고리즘을 이용한 LZ77 알고리즘의 성능 개선

(Performance Improvement of LZ77 Algorithm using a Strategy Table and a Genetic Algorithm)

정 순 철 † 서 동 일 † 문 병 로 ††
(Soonchul Jung) (Dong-Il Seo) (Byung-Ro Moon)

요 약 저장 공간이나 전송 시간을 줄여서 비용을 아끼는 데이타 압축 기술은 그 유용성 때문에 오래 전부터 연구되어 왔다. Lempel-Ziv 77(LZ77) 알고리즘은 실용적인 사전-기반 비손실 압축 알고리즘이다. 기존의 LZ77 알고리즘에서 알고리즘의 성능에 큰 영향을 미치는, 사전의 크기는 고정되어 있다. 본 논문에서는 사전의 크기를 동적으로 바꾸면서 압축을 하는 동적 LZ77 알고리즘과 동적 LZ77 알고리즘에서 사용하는 전략을 진화시키는 유전 알고리즘을 소개한다. 유전 알고리즘으로 진화시킨 전략을 가지고 동적 LZ77 알고리즘은 기존의 LZ77 알고리즘보다 최대 약 16%까지 더 좋은 압축 효율을 보여 주었다.

키워드 : 압축 알고리즘, 유전 알고리즘, Lempel-Ziv

Abstract Data compression techniques have been studied for decades because they saved space and time to reduce costs. The Lempel-Ziv 77 (LZ77) is a dictionary-based, lossless compression algorithm. The dictionary size of the LZ77 algorithm is fixed, and the performance of the algorithm is highly dependent on its dictionary size. In this paper, we suggest a dynamic LZ77 algorithm that changes its dictionary size during compression, and also we suggest a genetic algorithm to evolve the dictionary-resizing strategies. The suggested algorithm outperformed the original version up to about 16%.

Key words : compression algorithm, genetic algorithm, Lempel-Ziv

1. 서 론

데이타 압축(Data Compression)이란 저장 공간이나 전송 시간을 줄이기 위해서 데이타의 크기를 줄이는 과정을 말한다[1]. 데이타 압축을 잘 이용하면 기억 장치나 통신 자원을 효율적으로 이용할 수 있어서 그 비용을 상당히 줄일 수 있다. 압축 기술은 데이타베이스, 파일 저장, 데이타 통신, 음성 및 화상 통신 등 컴퓨터의 거의 모든 분야에서 사용되고 있다. 특히 미래에는 음성 및 화상 데이타를 처리하는 분야에서 압축 기술은 중요한 역할을 담당할 것이다. 데이타 압축은 크게 두 가지 종류로 나뉠 수 있다. 하나는 압축된 데이타로 원본 데이타를 완전하게 복원할 수 있는 비손실 압축이고, 다른 하나는 원본 데이타를 근사적으로 복원하는 손실 압축이다. 비손실 압축은 주로 파일 보관, 데이타 통신 분야에서 이용되고, 손실 압축은 영상, 음성 같은 멀티미디어 데이타를 저장할 때 쓰인다.

Lempel-Ziv 77(LZ77) 알고리즘[2]은 1977년 Lempel과 Ziv에 의해서 발표된, 사전-기반 비손실 압축 알고리즘이다. 이전에 출현했던 문자열들로 구성된 사전을 구축하고, 현재 문자열과 일치되는, 사전안 내용의 위치와 길이를 저장함으로써 압축을 수행하게 된다. LZSS[3], LZW[4], LZH[5] 같은 알고리즘들은 모두 LZ77 알고리즘을 기초로 파생된 것들로서, 지금도 새로운 LZ-계열 알고리즘[6][7]들이 계속 발표되고 있다. 현재 유닉스 compress, GNU gzip, MS-DOS ARC, LHA, PKZIP, ZOO 등이 모두 LZ-계열 압축 알고리즘을 사용하는 프로그램들이다.

유전 알고리즘(Genetic Algorithm: GA)은 진화의 원리에 기반한 확률적 문제 공간 탐색 도구로서 1976년 Holland에 의해 소개되었다[8]. 자연계의 각 개체들이

† 비 회 원 : 서울대학교 전기컴퓨터공학부
samuel@soar.snu.ac.kr
diseo@soar.snu.ac.kr
†† 정 회 원 : 서울대학교 컴퓨터공학부 교수
moon@soar.snu.ac.kr
논문접수 : 2003년 12월 31일
심사완료 : 2004년 9월 22일

교차, 변이를 통해서 진화하듯이, 유전 알고리즘에서는 교차, 변이를 모방한 연산으로 문제 공간을 탐색한다. 문제의 해는 유전 알고리즘에서 하나의 염색체에 대응된다. 유전 알고리즘은 기존의 일반적인 탐색 알고리즘과는 달리 하나의 해를 다루는 것이 아니라 집단으로 해들을 유지한다. 유전 알고리즘은 가스 파이프 라인의 최적화, 순회 판매원 문제, 그래프 분할, 이미지 압축, 로봇의 행동 진화, 신경망 학습, 전자상거래[9-11] 등 다양한 분야에서 사용되고 있다.

Ng와 Ravishankar는 사전 기반 압축 알고리즘에서 쓰이는 사전의 내용을, 유전 알고리즘을 이용하여 최적화하려고 시도하였다[12]. 그들은 허프만 트리를 이용해서 구축한 사전보다 더 좋은 압축율을 보여주는 사전을 유전 알고리즘이 만들었다고 보고하였다. LZ77 알고리즘에서 중요한 매개 변수는 사전의 크기를 결정하는 변수이다. 사전의 크기는 알고리즘의 성능에 큰 영향을 미친다. 압축할 데이타의 특성에 따라 사전의 크기를 동적으로 바꿀 수 있다면 알고리즘의 성능을 크게 향상시킬 수 있을 것이다. 본 논문에서는 LZ77 알고리즘의 고정 매개 변수를 압축 동안에 동적으로 변화시킬 수 있는 버전을 제시하고, 그 변화 전략을 진화시키는 유전 알고리즘을 제안한다.

2절에서는 기존의 LZ77 알고리즘과 새로운 동적 LZ77 알고리즘에 대하여 자세하게 설명한다. 3절에서는 본 논문에서 사용할 유전 알고리즘과 그것의 구성 요소들에 대하여 설명한다. 4절에서는 제안된 유전 알고리즘에 대한 실험 결과를 보여주고, 마지막 절에서 결론을 맺는다.

2. LZ77 알고리즘

본 논문에서, 압축이 될 입력 데이타를 메시지(message)라고 부르기로 한다. α개 기호들의 집합 $A=\{a_1, a_2, \ldots, a_\alpha\}$가 있을 때, 메시지는 A의 기호로 이루어진 문자열 $s_1 s_2 s_3 \ldots s_k$ ($s_i \in A$)이다. 편의상 $\alpha=256$이라고 하자. 이때 하나의 기호는 8비트 공간을 차지한다. $bit_n(n')$은 숫자 n'을 n 비트로 표현한 비트열을 나타낸다. 예를 들어, $bit_8(64)$는 비트열 01000000을 나타낸다.

LZ77 알고리즘은 가변 길이의 문자열에 대응되는 고정 길이의 부호를 저장한다. LZ77 알고리즘은 슬라이딩 윈도우(sliding window)라는 것을 사용한다. 윈도우는 두 부분으로 쪼개지는데, 앞부분은 이미 압축된 메시지 부분으로서 사전이라고 불리고, 뒷부분은 압축될 메시지 부분으로 미리보기 버퍼라고 불린다. 미리보기 버퍼의 시작 위치를 커서라고 한다. 사전의 크기를 $M=2^m$, 미리보기 버퍼의 크기를 $L=2^l$이라고 하자. m과 l은 프로그램의 매개변수로서, 기존 정적 LZ77 알고리즘에서는 실행 동안에 고정되어 있는 값이다. 즉 사전과 미리보기 버퍼의 크기는 고정되어 있다.

2.1 정적 LZ77 알고리즘

다음은 기존 LZ77 알고리즘의 부호화 과정을 기술하고 있다.

1. 압축을 시작하기 전, 사전은 특정 기호(a_1)로 채워져 있고, 커서는 메시지의 시작 위치를 가리킨다.
2. 커서에서 시작하는, 길이 $l'(1 \le l' \le L)$인 문자열이 사전에 있는지 조사한다. 가장 큰 l'을 가진 문자열에 일치하는, 사전 안의 문자열의 위치(커서에서 떨어진 거리)를 $m'(0 \le m' < M)$이라고 하자. 만약 그런 문자열이 존재하지 않는다면, $m'=0, l'=0$으로 저장한다.
3. c를 미리보기 버퍼에서, 최장 일치 문자열 다음의 기호($l'+1$번째 기호)라 하자. 부호어 (m', l', c)를 출력한다. $m'>0$이면, $\{ bit_m(m'), bit_l(l'-1), bit_8(c) \}$로 이루어진 비트열을 출력한다. 실제 일치된 길이(l')보다 1만큼 작은 숫자를 출력함을 주의한다. 출력되는 비트 개수가 $m+l+8$ 임은 쉽게 알 수 있다. 압축되는 메시지의 길이는 $(l'+1)\cdot 8$ 비트이므로 실제적으로 이 단계에서 $(l'+1)\cdot 8-(m+l+8)=l'\cdot 8-(m+l)$ 비트만큼 이득을 얻는다. $m'=0$의 경우, $\{ bit_m(m'), bit_8(c) \}$를 출력한다.
4. 커서의 위치를 $l'+1$만큼 뒤로 옮김으로써, 윈도우를 슬라이드시킨다. 커서가 메시지의 끝에 도달했다면, 부호화를 종료하고, 그렇지 않으면, 단계 2로 간다.

그림 1은 메시지 "aabaababcaabab"를 압축하는 LZ77 알고리즘의 부호화 예제이다.

Step	Message	Output Code
1	[a] *a b a* a b a b c a a b a b	(0, 0, a)
2	a [a] *b a a* b a b c a a b a b	(1, 1, b)
3	a a b [a] *a b a* b c a a b a b	(3, 3, a)
4	a a b a a b a [b] *c a a* b a b	(2, 1, c)
5	a a b a a b a b c [a] *a b a* b	(6, 4, b)

그림 1 M=8, L=4로 설정된 LZ77 알고리즘의 예제. 상자로 둘러 싸인 기호는 커서를 나타낸다. 사전의 기호들은 볼드체로, 미리보기 버퍼의 기호들은 이탤릭체로 쓰여졌다.

다음은 LZ77 복호화 과정을 기술하고 있다.

1. 복호하기 전, 사전은 특정 기호(a_1)로 채워져 있고, 커서의 위치는 0이다.
2. 적절히 부호어를 읽어 들여, m', l', c 를 얻는다.
3. $m'>0$이면, 사전의 해당 위치에서 l' 길이만큼을 현재 커서의 위치에 복사한다.

4. c를 다음 위치에 복사한다.
5. 커서의 위치를 $l'+1$ 만큼 뒤로 옮김으로써, 윈도우를 슬라이드시킨다. 더 이상 복호할 부호어가 없다면 종료하고, 그렇지 않으면 단계 2로 간다.

그림 2는 그림 1에서 부호화했던 부호어들을 복호하여 원본 메시지 "aabaababcaabab"를 복원하는 예제이다.

Input Code	Output Message
(0, 0, a)	a □
(1, 1, b)	a a b □
(3, 3, a)	a a b a a b a □
(2, 1, c)	a a b a a b a b c □
(6, 4, b)	a a b a a b a b c a a b a b

그림 2 LZ77 알고리즘을 이용해서 부호화된 부호어들을 복호하는 예제. 밑줄이 그어진 기호들은 현재 커서 위치에서 다시 출현하는 기호들로서, 다음 단계에서 c를 쓰기 전에 복사된다.

어떤 메시지 S와 S가 압축된 후의 문자열 S'이 주어졌을 때, 압축효율을 $CE_s = \frac{S\text{의 길이}}{S'\text{의 길이}}$로 나타낼 수 있다. 그림 1의 경우, 부호어 하나는 보통 $m+l+8 = 3+2+8 = 13$ 비트를 차지하므로 대략적인 압축효율은 $\frac{14 \cdot 8}{13 \cdot 5} \fallingdotseq 1.72$이다. 압축 효율이 클수록 압축 알고리즘의 성능은 더 좋다고 말할 수 있다.

슬라이딩 윈도우의 크기는 LZ77 알고리즘의 성능에 영향을 미친다. 윈도우가 크면, 그렇게 하지 않았다면 찾을 수 없었을 단어들을 발견할 가능성이 더 높기 때문에 압축 효율을 높일 수 있다. 그러나, 윈도우가 커질수록 부호어 자체의 크기($m+l+8$)도 덩달아 커지기 때문에 적절한 크기를 유지해야 한다. 또한 각 메시지마다 그것의 특성, 즉 텍스트 파일인지, 이진 파일인지 등등에 따라서 적절한 윈도우의 크기는 달라진다. 그리고 tar 파일처럼 그 안에 다양한 특성을 동시에 가진 메시지도 있을 수 있다. 이런 종류의 메시지들은 압축이 되는 동안에 최적의 윈도우 크기가 그때 그때마다 달라질 것이다.

2.2 동적 LZ77 알고리즘

본 절에서는 메시지를 압축하는 동안 동적으로 슬라이딩 윈도우의 크기를 바꾸는 알고리즘을 제안한다. 슬라이딩 윈도우의 크기를 바꾼다는 것은 m과 l의 값을 조정한다는 뜻이다. m의 값이 1만큼 더 커진다면, M의 값, 즉 사전의 크기는 두배가 된다. 동적 LZ77 알고리즘의 기본적인 아이디어는 주기적으로 과거의 m', l'들의 통계를 보고, 현재의 m, l의 값을 전략 테이블에 따라 조정하는 것이다.

$$\begin{vmatrix} u_{1,1} & \cdots & u_{1,C} & v_{1,1} & \cdots & v_{1,C} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ u_{C,1} & \cdots & u_{C,C} & v_{C,1} & \cdots & v_{C,C} \end{vmatrix}$$

$$(u_{i,j}, v_{i,j} \in \{-1, 0, 1\}, C > 0)$$

전략 테이블은 위와 같이 -1, 0, 1 값을 갖는 요소로 이루어진 행렬이다. C는 전략의 정밀도를 나타낸다. C의 값이 클수록 과거의 m', l'의 통계 값들을 세밀하게 분류하게 된다. 전략 테이블과 함께 주기 P, 초기의 m, l값인 m_{ini}, l_{ini}, m, l의 하한과 상한인 $m_{\min}, l_{\min}$, $m_{\max}, l_{\max}$가 정의되어 있을 때, 동적 LZ77 알고리즘의 부호화 과정은 다음과 같다.

1. m_{ini}, l_{ini}를 가지고 P 개의 부호어를 생성한다.
2. m_{cur}, l_{cur}를, 바로 전 P 개의 부호어를 만들 때 이용되었던 m, l이라고 하고, $a_{m'}, a_{l'}$를 각각 과거 P 개의 부호어의 m', l'들의 평균값이라고 하자. 이들을 이용해서 다음 번 P 개의 부호어를 부호화할 때 사용할 새로운 m과 l 값인 $m_{\text{new}}, l_{\text{new}}$를 아래 식으로 구한다.

$$i = \lfloor \frac{a_{m'}}{2^{m_{cur}}} \times C \rfloor, \quad j = \lfloor \frac{a_{l'}}{2^{l_{cur}}} \times C \rfloor,$$

$$m_{\text{new}} = \max(\ \min(m_{cur} + u_{i,j}, m_{\max})\ ,\ m_{\min}\ )$$

$$l_{\text{new}} = \max(\ \min(l_{cur} + v_{i,j}, l_{\max})\ ,\ l_{\min}\ )$$

3. P 개의 부호어를 생성하고, 압축할 것이 남아 있다면 단계 2로 간다.

정적 LZ77 알고리즘과 비슷한 방법으로 복호화를 할 수 있으므로, 이에 대한 설명은 생략한다.

동적 LZ77 알고리즘의 성능은 전략 테이블의 내용에 따라 많이 달라진다. $u_{i,j}, v_{i,j}$의 값이 모두 0이라면, 기존의 정적 LZ77 알고리즘과 동일함을 알 수 있다. 문제 공간의 크기라고 할 수 있는, 가능한 전략 테이블의 개수는 $3^{2 \cdot C^2}$으로, C가 조금만 커져도 컴퓨터로 모든 전략 테이블에 대한 동적 LZ77 알고리즘의 압축효율을 일일이 조사하는 게 불가능해진다. 본 논문에서는 유전 알고리즘으로 전략 테이블을 진화시키는 방법을 제안한다.

3. 동적 LZ77 알고리즘을 위한 유전 알고리즘

본 논문에서는 다음과 같은 전형적인 안정-상태 유전 알고리즘을 사용한다. 새로운 해를 만들자마자 바로 대치시키는 안정-상태 유전 알고리즘은, 해들을 집단으로 교배, 변이 시키는 세대 유전 알고리즘보다 수렴이 빠르고, 구현이 쉽다.

```
안정상태 유전 알고리즘 {
    해집단을 초기화한다.
    while( 종료 조건이 만족되지 않는다면 ) {
        해집단에서 부모1, 부모2를 선택한다.
        부모1, 부모2를 교배하여 자식해를 만든다.
        if( 변이를 할 것인가 )
            자식해를 변이시킨다.
        해집단의 한 개체를 자식해로 대치시킨다.
    }
    가장 좋은 해를 반환한다.
}
```

유전 알고리즘은 우선 해집단을 초기화하고, 선택, 교배, 변이 및 대치 연산을 반복하게 된다. 유전 알고리즘을 설계할 때, 이들 연산자 외에도 염색체(해)를 어떻게 표현할 것인지가 중요한 요소가 된다.

3.1 염색체 표현

본 논문에서는 전략 테이블 하나가 한 개의 염색체가 된다. $u_{i,j}, v_{i,j}$는 전략 테이블의 요소로서, 과거 m', l'의 평균값에 의해서 특정 $u_{i,j}, v_{i,j}$가 선택되고, 선택된 $u_{i,j}, v_{i,j}$의 값에 따라 현재의 m, l의 값이 1만큼 늘어나거나, 그대로 유지되거나, 또는 1만큼 줄어들게 된다. 결과적으로 전략 테이블의 $u_{i,j}, v_{i,j}$의 값이 동적 LZ77 알고리즘의 압축 효율에 큰 영향을 미친다. 염색체는 그림 3처럼 $(u_{i,j}, v_{i,j})$쌍을 row-major로 하여, 이차원 전략 테이블을 변환시킴으로써 만들어진, $2 \cdot C^2$개의 유전자로 이루어진 일차원 배열이다. 본 논문에서는 $C=8$로 설정되었다. 그러므로 염색체의 길이는 128이 된다. 염색체가 얼마나 우수한 지를 나타내는 적합도는 다음과 같이 계산된다. 주어진 훈련 메시지 $S_1, \ldots, S_k$ 각각을 염색체가 표현하는 전략 테이블을 가지고 압축했을 때, 훈련 메시지들의 압축 효율의 평균인 $\frac{\sum CE_{S_i}}{k}$가 해당 염색체의 적합도가 된다.

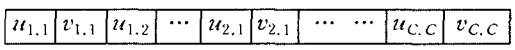

그림 3 염색체 구조

3.2 초기화

초기화는 유전 알고리즘을 시작할 때, 문제 공간안에서 해들을 어떤 식으로 배치할 지 결정하는 과정이다. 본 논문에서는 해집단의 각 해의 각 유전자에 대해 -1, 0, 1 중에서 임의의 값을 설정함으로써 초기화가 된다. 즉 문제 공간안에서 해집단의 해들은 무작위로 분포하게 된다.

3.3 선택 연산자

선택 연산은 교배 연산에 쓰일, 해집단 안의 해 2개를 뽑는 과정으로서, 그 2개의 해를 부모해(parent)라고 부른다. 선택 연산을 할 때, 확률적으로 보다 우수한 해들이 더 많은 자손을 남기도록 선택되어야 한다. 그렇게 함으로써, 문제 공간안에서 우수한 해의 근방이 더 많이 조사되는 효과가 있다. 본 논문에서는 선택 연산자로서 해집단에서 가장 좋은 해가 가장 나쁜 해보다 뽑힐 확률이 p배 크도록 조정하여 부모해를 뽑는 적합도 비례 룰렛휠 선택 연산자가 사용된다[13]. 본 논문에서는 $p=5$로 하였다. 주어진 문제에서 해의 선택 확률을 조정하지 않고 그대로 사용할 경우, 해집단에서 가장 좋은 해와 나쁜 해의 확률 차이가 너무 커서, 나쁜 해들은 거의 선택될 기회를 갖지 못한다. 결과적으로 해의 다양성을 급속히 떨어뜨려 해집단을 국지 공간으로 몰아버림으로써, 성능을 떨어뜨리게 된다.

3.4 교배 연산자

교배 연산은 2개의 부모해에 들어있는 유전자들을 교대로 물려받아 결합하여 새로운 자식해를 생성하는 과정으로서, 집단내 해들의 유용한 정보를 공유하기 위한 방법이다. 본 논문에서는 교배 연산자로서 3점 교배 연산자가 사용되었다. 그림 4처럼 이 연산자는 두 부모해를 같은 위치에서 각각 4부분으로 조각내서, 부모들로부터 교대로 유전자를 물려받도록 해서 새로운 해(자식해, offspring)를 생성한다. 자름선이 3개, 염색체의 길이가 128이므로, 특정 부모해에서 약 $_{127}C_3$개의 서로 다른 자식해를 만들어 낼 수 있다.

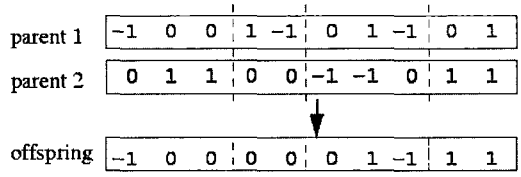

그림 4 3점 교배 연산의 예

3.5 변이 연산자

변이 연산은 부모해에 없는 속성을 도입하여 탐색 공간을 넓히려는 목적을 가진 연산으로서, 염색체내의 일부 유전자들을 무작위로 바꿈으로써 이루어진다. 변이 연산은 임의로 일어나므로 일반적으로 해의 품질을 떨어뜨린다. 그러나 변이의 결과로 잘못 만들어진 유전자는 곧 도태되며, 가끔 발생하는 성공적인 변이의 경우에는 해집단의 전체적인 품질을 향상시키는 데 기여를 하게 된다. 본 논문에서는 그림 5처럼 염색체내의 각 유전자에 대해서 0.05의 확률로 -1, 0, 1 중에서 임의의 값으로 설정한다. 염색체의 길이가 128이므로, 염색체가 변이되는 경우, 평균 4.27($=128 \cdot 0.05 \cdot (2/3)$)개의 유전자가 변이된다.

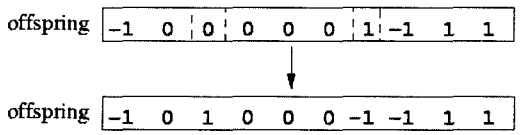

그림 5 변이 연산의 예

3.6 대치 연산자

대치 연산은 새로 생긴 자식해를 해집단에 편입시키기 위한 과정으로서, 해집단에서 도태될 해를 하나 찾아서 자식해와 바꾸는 연산을 한다. 대치 연산은 선택 연산과 마찬가지로 해집단의 다양성과 관련이 있다. 본 논문에서 하는 대치 연산의 방법은 다음과 같다. 부모1의 적합도가 자식해보다 낮다면, 부모1이 자식해로 대치된다. 그렇지 않다면 부모2와 비교하여 부모2의 적합도가 자식해보다 낮다면, 부모2가 자식해로 대치된다. 자식해가 두 부모 모두보다 나쁘다면, 해집단내에서 가장 적합도가 낮은 해가 자식해로 대치된다.

3.7 종료 조건

유전 알고리즘을 종료하기 위한 조건은 문제에 따라 다양하다. 본 논문에서는 유전 알고리즘으로 직접 파일을 압축하는 것이 아니라 동적 LZ77 알고리즘의 전략을 결정하는 것이기 때문에 동적 LZ77 알고리즘의 실제 압축 시간에는 영향을 미치지 않는다. 따라서 유전 알고리즘의 실행 시간은 큰 문제가 되지 않는다. 전략을 충분히 진화시키기 위해서, 1000세대동안 계속해서 현재 가장 좋은 해보다 더 좋은 해가 나오지 않을 때 유전 알고리즘이 종료되도록 하였다.

4. 실험 결과

본 논문의 모든 프로그램들은 C++로 작성되었고, RedHat Linux 8.0의 Intel Pentium III 1GHz 상에서 실험이 진행되었다. 표 1에 실험에 사용될 벤치마크 파일들이 나열되어 있다. binary 형식의 벤치마크 파일로서 RedHat[1] 8.0 운영체제의 실행 파일들을 선택하였고, text 형식의 파일들은 Calgary Corpus[14]로부터 수집되었다. 이 파일들을 사용하여 모두 네가지 실험을 진행하였다. 첫 번째 실험은 벤치마크 파일들에 대해서 m과 l의 변화에 따른, 기존의 정적 LZ77 알고리즘의 성능 변화를 보기 위한 것이다. 두 번째 실험에서는 binary 형식의 파일들을 보다 잘 압축하도록, 유전 알고리즘으로 동적 LZ77 알고리즘의 전략을 진화시키고 나서, 그것의 성능을 정적 LZ77 알고리즘과 비교한다. 세 번째 실험은 text 형식의 파일들에 대해서 같은 실험을 한다. 마지막으로 binary, text 두 형식 모두가 포함된 파일들에 대해서 실험을 한다.

1) http://www.redhat.com

표 1 실험에 사용될 훈련용, 테스트용 파일들

파일 이름	출처	속성	크기
gcc	RedHat	binary	77,996
tcsh	RedHat	binary	365,432
xterm	RedHat	binary	272,119
gdb	RedHat	binary	2,230,756
bash	RedHat	binary	626,188
xman	RedHat	binary	58,405
book1	Calgary	text	768,771
book2	Calgary	text	610,856
paper1	Calgary	text	53,161
paper2	Calgary	text	82,199
paper3	Calgary	text	46,526
paper4	Calgary	text	13,286

사전 검사를 통해 동적 LZ77 알고리즘의 매개 변수들은 $m_{ini}=10, l_{ini}=4,\ m_{\min}=0, l_{\min}=0,\ m_{\max}=19, l_{\max}=19,$ $P=40$으로 설정되었다. 해집단의 크기는 200, 변이 연산이 자식해에 적용될 확률은 0.1로 각각 설정되었다. 모든 실험은 50번씩 실행되었다.

4.1 정적 LZ77 알고리즘의 성능에 관한 실험 결과

정적 LZ77 알고리즘의 매개 변수 m과 l의 적당한 값은 보통 각각 13~15와 3~5로 알려져 있다. 본 절에서는 벤치마크 파일들에 대해서 최적의 압축효율을 내는, 정적 LZ77 알고리즘의 매개 변수를 알아내기 위해서, m과 l의 값을 계속 바꿔가면서 압축을 한다.

표 2는 gdb, bash, xman, book2, paper3, paper4 파일에 대한 정적 LZ77 알고리즘들의 압축 효율을 보여준다. LZ7712,3을 $m=12, l=3$으로 설정된 정적 LZ77 알고리즘이라고 하자.

각 파일에 대해서 가장 좋은 결과를 보인 알고리즘의 압축 효율은 볼드체로 쓰여졌다. binary 형식의 파일에 대해서는 $LZ77_{16,4}$가 다른 설정의 알고리즘들보다 평균적으로 좋은 결과를 보여 주었다. text 형식의 파일에 대해서는 $LZ77_{14,4}$가 좋았다. 전체 파일을 대상으로 했을 때는 $LZ77_{14,4}$와 $LZ77_{16,4}$ 둘 다 각각 3개씩 서로 앞선 결과를 보여 주고 있다. 각 파일에 대한 두 알고리즘의 압축 효율간의 차이를 고려하면 $LZ77_{16,4}$가 가장 좋다고 할 수 있다. 흥미롭게도 각 파일에 대한 최고의 압축 효율을 보여주는 알고리즘 중에는, 한 개 파일을 제외하고는, $LZ77_{14,4}$나 $LZ77_{16,4}$가 없었고, 이들은 대부분의 파일에 대해서 중 또는 상의 압축 효율을 보여 주었다. 한편 각 파일에 대한 최고의 압축 효율을 보여주는 알고리즘들이 서로 다르다는 것은 최적의, 전역적인 m과 l의 값은 존재하지 않음을 뜻한다.

표 2 m과 l의 변화에 따른 정적 LZ77 알고리즘의 압축 효율 변화

	gdb	bash	xman	book2	paper3	paper4
$LZ77_{12,3}$	1.604	1.561	1.569	1.803	1.715	1.673
$LZ77_{12,4}$	1.659	1.592	**1.595**	1.844	1.714	1.691
$LZ77_{12,5}$	1.629	1.554	1.553	1.795	1.655	1.635
$LZ77_{13,3}$	1.625	1.581	1.552	1.887	1.779	1.672
$LZ77_{13,4}$	1.696	1.625	1.588	1.963	1.795	**1.712**
$LZ77_{13,5}$	1.671	1.591	1.549	1.919	1.739	1.660
$LZ77_{14,3}$	1.637	1.592	1.531	1.956	1.816	1.629
$LZ77_{14,4}$	1.727	1.649	1.574	2.073	1.851	1.670
$LZ77_{14,5}$	1.707	1.619	1.539	2.036	1.796	1.622
$LZ77_{15,3}$	1.644	1.597	1.490	2.006	1.807	1.566
$LZ77_{15,4}$	1.751	1.668	1.535	2.174	**1.857**	1.608
$LZ77_{15,5}$	1.736	1.641	1.504	2.144	1.806	1.564
$LZ77_{16,3}$	1.645	1.595	1.478	2.039	1.748	1.508
$LZ77_{16,4}$	1.767	**1.677**	1.544	2.258	1.803	1.551
$LZ77_{16,5}$	1.758	1.655	1.519	2.239	1.755	1.510
$LZ77_{17,3}$	1.640	1.585	1.426	2.044	1.685	1.454
$LZ77_{17,4}$	1.775	1.675	1.490	2.314	1.740	1.497
$LZ77_{17,5}$	1.771	1.657	1.469	2.306	1.696	1.459
$LZ77_{18,3}$	1.628	1.559	1.376	2.024	1.627	1.404
$LZ77_{18,4}$	**1.776**	1.656	1.441	**2.332**	1.682	1.447
$LZ77_{18,5}$	**1.776**	1.640	1.421	**2.332**	1.642	1.412

이 결과에 따라 앞으로 binary 형식 파일에 대한 실험의 경우 $LZ77_{16,4}$, text 형식 파일에 대한 실험의 경우 $LZ77_{14,4}$, 두 형식 파일들을 혼용한 실험의 경우 $LZ77_{16,4}$와 비교하기로 한다.

4.2 binary 형식의 파일들에 대해서 훈련된 전략을 가진 동적 LZ77 알고리즘의 성능

binary 형식의 파일들을 대상으로, 유전 알고리즘을 이용하여 동적 LZ77 알고리즘의 전략을 진화시키고 나서, 같은 형식의 파일들을 압축했을 때의 결과에 대해 알아본다. gcc, tcsh, xterm 파일이 염색체의 적합도 측정을 위한 훈련용 파일로 선택되었다. 이 실험에서 사용된 유전 알고리즘을 GA_{binary}라 하자.

그림 6은 세대에 따른 해집단의 평균 적합도와 최고 적합도의 변화를 보여준다. 약 7000세대 이후로 해집단의 평균 적합도와 최고 적합도사이에 거의 차이가 없음을 알 수 있다. 7000세대 정도에 해집단이 수렴했음을 예상할 수 있다. 평균 유전 알고리즘의 실행 시간은 약 20,000초였다.

표 3에 gdb, bash, xterm 파일에 대한 동적 LZ77 알고리즘의 결과가 $LZ77_{16,4}$와 비교된다. GA_{binary}-Avg는, 50번의 실험으로 산출된 전략들을 가지고 행한, 동적 LZ77 알고리즘의 평균 결과이다. GA_{binary}-Best는 50개의 해중에 가장 적합도가 높았던 전략을 이용했을 때의, 동적 LZ77 알고리즘의 결과이다. 어떤 전략의 적합도가 가장 높았다고 하더라도 실제 적용시에는 다른 전략보다 압축 효율이 나빠질 수도 있음을 유의해야 한다. 3개 파일 모두에 대해서 $LZ77_{16,4}$보다 GA_{binary}-Avg, GA_{binary}-Best의 압축 효율이 더 좋음을 알 수 있다. GA_{binary}-

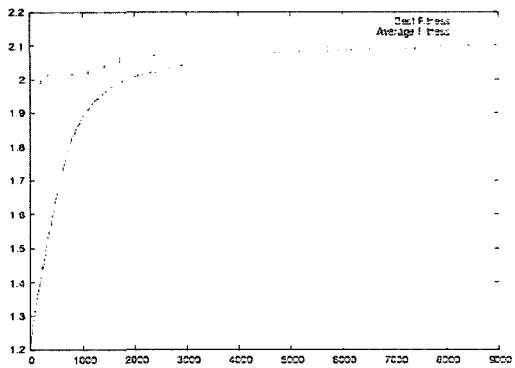

그림 6 세대에 따른 해집단의 평균 적합도와 최고 적합도의 변화

표 3 binary 형식의 파일을 대상으로 훈련된 전략을 이용한 동적 LZ77 알고리즘의 각 파일에 대한 압축 효율

	gdb	bash	xman
$LZ77_{16,4}$	1.767	1.677	1.544
GA_{binary}-Avg	1.783	1.717	1.627
GA_{binary}-Best	**1.840**	**1.739**	**1.662**

Avg는 $LZ77_{16,4}$보다 약 1~5%까지 성능이 좋았다. GA_{binary}-Best는 $LZ77_{16,4}$보다 약 4~8%까지 성능이 좋았다. $LZ77_{16,4}$는 3개 파일을 압축하는 데에 총 7.9초가 걸렸고, GA_{binary}-Best는 약 8.6초가 걸렸다.

4.3 text 형식의 파일들에 대해서 훈련된 전략을 가진 동적 LZ77 알고리즘의 성능

book1, paper1, paper2 파일이 염색체의 적합도 측정을 위한 훈련용 파일로 선택되었다. 이 실험에서 사용된 유전 알고리즘을 GA_{text}라 하자. 표 4에서 book2, paper3, paper4 파일에 대한 동적 LZ77 알고리즘의 결과가 $LZ77_{14,4}$와 비교되었다. 두 번째 실험 결과와 비슷하게 GA_{text}-Avg는 $LZ77_{14,4}$보다 약 3~15%까지 성능이 좋았다. GA_{text}-Best는 $LZ77_{14,4}$보다 약 4~16%까지 성능이 좋았다. 또한 표 3과 4를 보면, GA_{binary}-Avg, GA_{text}-Avg보다 각각 GA_{binary}-Best, GA_{text}-Best의 결과가 일관성있게 더 좋게 나온다는 것은 전략이 충실하게 진화되었음을 알려준다.

표 4 text 형식의 파일을 대상으로 훈련된 전략을 이용한 동적 LZ77 알고리즘의 각 파일에 대한 압축 효율

	book2	paper3	paper4
$LZ77_{14,4}$	2.073	1.851	1.670
GA_{text}-Avg	2.382	1.900	1.756
GA_{text}-Best	**2.403**	**1.917**	**1.770**

4.4 binary, text 형식의 파일들이 혼용되어서 훈련된 전략을 가진 동적 LZ77 알고리즘의 성능

binary 형식 파일, text 형식 파일 둘 모두를 이용하여 전략을 진화시키는 실험을 한다. 이런 종류의 훈련은 형식에 상관없이 파일을 잘 압축시키는 범용적인 전략을 만들 수 있을 것이라 기대된다. gcc, tcsh, xterm, book1, paper1, paper2 파일들이 훈련용 파일로 선택되었다. 이 실험에서 사용된 유전 알고리즘을 GA_{all}이라 하자.

표 5에서 gdb, bash, xman, book2, paper3, paper4 파일에 대한 동적 LZ77 알고리즘의 결과가 $LZ77_{16,4}$와 비교되었다. gdb 파일을 제외하고는 정적 LZ77 알고리즘보다 동적 LZ77 알고리즘의 성능이 2~14%까지 좋았다.

GA_{all}-Avg와 GA_{binary}-Avg, GA_{text}-Avg를 비교했을 때 각 테스트 파일마다 결과가 엇갈리게 나왔다. GA_{binary}, GA_{text}의 평균 종료 세대는 각각 7192, 7109이었고, 훈련용 파일의 개수는 3개였고, GA_{all}의 평균 종료 세대는 8300, 훈련용 파일 개수는 6개였음을 고려한다면, GA_{all}이 훨씬 길게 실행되어 전략이 진화되었음을 알 수 있다. 그 결과로 GA_{all}-Avg와 GA_{binary}-Avg, GA_{text}-Avg의 전략들이 평준화되었기 때문에 결과들이 서로 엇갈리게 나온 것으로 추측된다. 그러나 GA_{all}-Best와 GA_{binary}-Best, GA_{text}-Best의 결과를 비교했을 때, 모든 테스트 파일에 대하여 GA_{binary}-Best, GA_{text}-Best가 좋았다.

표 3, 4, 5에 나온 결과들을 종합해보면, 유전 알고리즘으로 특정 형식의 파일에 대해서 훈련이 된 전략을 가진 동적 LZ77 알고리즘으로 같은 형식의 파일들을 압축하는 것이 가장 좋은 성능을 보이는 것 같다.

그림 7에 GA_{all}-Best의 전략을 보여준다. +1은 '+'로, -1은 '-'로, 0은 공백으로 표현되어 있다. 전략을 보면 전반적으로 +와 -가 무작위로 분포되어 있지도 않지만 완전히 편향되게 배열되어 있지도 않은 것으로 보아, m 또는 l의 값이 무조건 크게 되거나 작게 되는 경우를 막기 위한 것으로 보인다. 그림에서 왼쪽에 있는 $u_{i,j}(1 \le i,j \le 8)$ 부분만 분석해 보면, 총 64개 요소 중에서 +1은 26개, 0은 23개, -1은 15개가 있었다. 전반적으로 m의 크기를 키우려는 경향이 있음을 알 수 있다. 특히 $u_{i,j}(5 \le i,j \le 8)$ 부분만 보면 전체 16개 중에서 +1이 8개가 될 정도로 많아서, 사전에서 매치되는 문자열의 거리가 커서로부터 멀고, 매치 길이가 길면 m이 쉽

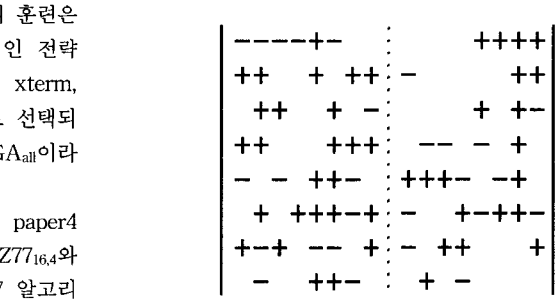

그림 7 GA_{all}-Best의 전략

표 5 binary, text 두 형식의 파일들을 대상으로 훈련된 전략을 이용한 동적 LZ77 알고리즘의 각 파일에 대한 압축 효율

	book2	paper3	paper4	gdb	bash	xman
$LZ77_{16,4}$	2.258	1.803	1.551	1.767	1.677	1.544
GA_{all}-Avg	2.394	1.889	1.748	**1.799**	1.712	1.635
GA_{all}-Best	**2.397**	**1.913**	**1.769**	1.609	**1.732**	**1.658**

게 커짐을 알 수 있다. 그림의 오른쪽에 있는 $v_{i,j}(1 \le i,j \le 8)$ 부분에서는 총 64개 요소 중에서 0이 33개, +1이 20개, -1이 11개가 있었다. 대략적으로 l의 값을 현상 유지시키는 측면이 있음을 알 수 있다. 그런데 $v_{i,j}(1 \le i \le 4, 5 \le j \le 8)$부분에는 +1이 9개나 있어서, 커서로부터 매치되는 문자열의 위치가 가깝고 매치 길이는 길면, l의 값이 커질 확률이 높아진다.

그림 8은 GAall-Best의 전략을 가지고 bash 파일을 동적 LZ77 알고리즘으로 압축했을 때, 매 주기(X축)마다 변화하는 m과 l의 값(Y축)을 보여 주고 있다. 대략 2700주기에서 압축이 끝난 것으로 보아 2700* P(=40) = 108000개의 부호어로 파일이 부호화되었음을 알 수 있다. m은 첫부분과 마지막 부분에서는 크게 진동을 했으나, 주로 16-19사이에서 머물렀다. l은 거의 4로 고정이 되었고, 2300주기 근처에서는 0으로 떨어졌다가 다시 4로 복귀했다. 자세히 보면 m의 값이 먼저 떨어지고 그 다음에 l의 값이 떨어졌음을 알 수 있다. 그 주기에서 매치되는 문자열들이 없어서 m의 값이 떨어지고, m이 l과 같아졌을 때, 이론적으로 l이 m보다 크면 손해가 되기 때문에, 결국 l의 값도 떨어진 것으로 풀이된다. 그림 9는 같은 전략으로 book2 파일을 압축했을 때의 결과이다. 압축을 시작할 때에 조정을 거쳐 m은 18-19, l은 4로 고정되었다. 그림 10은 bash 파일 뒷부분에 book2 파일을 접합시켜서 만든 약 1.2M 바이트의 파일(bash+book2)을 압축시켰을 때의 결과를 보여준다. 이 파일은 내부적으로 앞부분은 binary 형식이었다가 뒷부분은 text 형식으로 바뀐다. bash+book2 파일의 앞부분과 bash 파일은 내용이 완전히 똑같기 때문에, 그림 10의 앞부분 패턴은 그림 8과 완전히 같다. 그림 10의 뒷부분 패턴도 그림 9와 거의 비슷하게 보인다. bash+book2 파일에서 book2 부분에 대해서 압축을 시작할 때 슬라이딩 윈도우의 사전에는 bash의 내용이 있

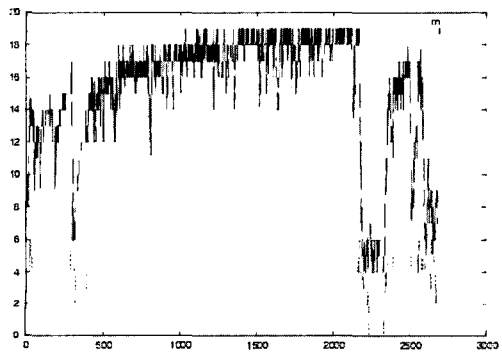

그림 8 bash 파일을 GA_{all}-Best의 전략을 가지고 압축을 할 때, m과 l 값의 변화 (X축은 주기)

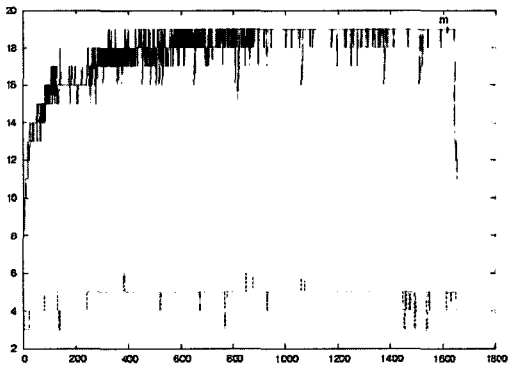

그림 9 book2 파일을 압축할 때의 m과 l 값의 변화

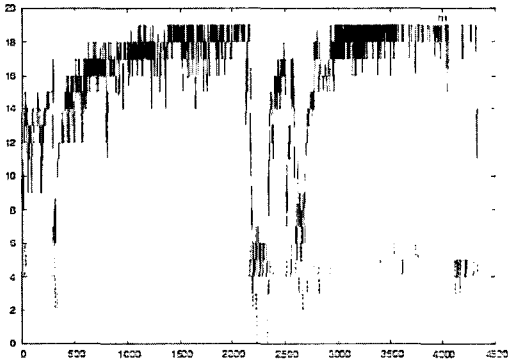

그림 10 bash+book2 파일을 압축할 때의 m과 l 값의 변화

기 때문에 거의 매치되는 문자열이 존재하지 않는다. 이 상황은 book2 파일을 압축할 때, 처음 사전에 특정 기호(a_1)만 가득 차 있는 상황과 비슷하다. 이러한 이유로 bash+book2 파일의 뒷부분과 book2 파일간의 m과 l의 변화 패턴이 비슷하게 된 것이라 추측된다. bash+book2 파일에 대한 압축 효율은 2.003으로서, bash 파일에 대한 압축 효율과 book2 파일에 대한 압축 효율의 중간 정도의 수치였다. 그림 10이 보여주는 결과는 파일 내부에서 형식이 갑자기 바뀌더라도 동적 LZ77 알고리즘의 성능이 떨어지지 않을 것임을 암시한다.

5. 결 론

본 논문에서는 동적으로 슬라이딩 윈도우의 크기를 바꾸는 LZ77 알고리즘을 위한 전략을 진화시키는 유전 알고리즘을 제안하였다. 제안된 유전 알고리즘으로 진화시킨 전략을 가지고 동적 LZ77 알고리즘은 기존의 LZ77 알고리즘보다 최대 약 16%까지 더 나은 압축효율을 보여주었다. text 형식의 파일들로 전략을 훈련시키는 것처럼, 특정 형식의 파일들만을 대상으로 훈련시

키고, 압축하는 것이 가장 좋은 성능을 나타냈다. bash+book2 파일의 압축 예에서 보듯이 동적 LZ77 알고리즘은 파일 내용의 급격한 변화에도 불구하고 견고한 성능을 보여 주었다.

현존하는 최신의 압축 프로그램들은 대부분 LZ77 알고리즘을 기초로 하여 작성되었다. 이들 프로그램에 본 논문에서 제안한 방법을 적용시킨다면 지금보다 더 좋은 성능을 보일 것이라 예상된다.

현재 전략 테이블은 이차원 형태의 행렬로 표현되지만, 염색체상에서는 일차원으로 변환되어 다루어지고 있다. 이런 이차원 형태의 데이타 구조를 일차원 선형 문자열로 변환되는 과정에서, 인접 정보와 같은 정보들이 손실된다고 알려져 있다[15]. 앞으로 염색체의 형태를 전략 테이블과 같은 이차원 형태로 하는 유전 알고리즘에 대하여 연구할 예정이다.

참 고 문 헌

[1] R. M. Gray, Source Coding Theory, Kluwer Academic Publishers, Boston, 1990.

[2] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Transactions on Information Theory, Vol. 23, pp. 337-342, 1977.

[3] J. A. Storer and T. G. Szymanski. "Data compression via textual substitution," Journal of ACM, Vol. 29, No. 4, pp. 928-951, 1982.

[4] T. A. Welch, "A technique for high-performance data compression," IEEE Computer, Vol. 17, No. 6, pp. 8-19, 1984.

[5] R. P. Brent, "A linear algorithm for data compression," Australian Computer Journal, Vol. 19, No. 2, pp. 64-68, 1987.

[6] C. Bloom, "LZP: a new data compression algorithm," IEEE Data Compression Conference, 1996.

[7] M. F. Oberhumer. "LZO - a real-time data compression library," URL: http://www.infosys.tuwien.ac.at/Staff/lux/marco/- lzo.html, 1997.

[8] J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.

[9] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[10] S. Jung and B. R. Moon, "Toward minimal restriction of genetic encoding and crossovers for the 2D Euclidean TSP," IEEE Trans. on Evolutionary Computation, Vol. 6, No. 6, 2002.

[11] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," IEEE Trans. on Computers, Vol. 45, No. 7, pp. 841-855, 1996.

[12] W. K. Ng and C. V. Ravishankar, "A preliminary study of genetic data compression," International Conference on Genetic Algorithms, pp. 566-573, 1995.

[13] 문병로, 유전 알고리즘, 다성 출판사, 2001.

[14] T. C. Bell, J. G. Cleary, and I. H. Witten, Text Compression, Prentice Hall, Englewood Cliffs, NJ, 1990.

[15] T. N. Bui and B. R. Moon, "Hyperplane synthesis for genetic algorithms," International Conference on Genetic Algorithms, pp. 102-109, 1993.

정 순 철

1998년 KAIST 전산학과 학사. 2000년 서울대 전산과학과 석사. 현재 서울대학교 전기 컴퓨터 공학부 박사과정. 관심분야는 유전 알고리즘, 순회 판매원 문제, 패턴 분석, 인공 신경망

서 동 일

1996년 KAIST 전산학과 학사. 1998년 KAIST 전산학과 석사. 현재 서울대학교 전기 컴퓨터 공학부 박사과정. 관심분야는 진화 알고리즘, 조합 최적화, 정보 이론, 알고리즘 설계

문 병 로

정보과학회논문지 : 소프트웨어 및 응용 제 31 권 제 8 호 참조