数理解析研究所講究録
第1554巻 2007年 243-249

部分文字列の高速復元に適した圧縮データ構造に関する研究

後藤 隆元 (Takamoto Goto) * 小野 廣隆 (Hirotaka Ono) †
定兼 邦彦 (Kunihiko Sadakane) † 山下 雅史 (Masafumi Yamashita) †
* 九州大学大学院システム情報科学府 † 九州大学大学院システム情報科学研究院
*†Department of Computer Science and Communication Engineering,
Graduate School of Information Science and Electrical Engineering, Kyushu University

1 はじめに

計算機上でサイズの大きなデータを扱う際，データサイズを小さくするために，元のデータに対して圧縮を行い，サイズを小さくした圧縮データを保存することがある．一般的に元のデータサイズが大きいほど圧縮率は向上するが，既存の大部分の圧縮アルゴリズムでは，その圧縮データから一部分だけを復元しようと思っても圧縮データを先頭から復元する必要があった．しかし，それでは復元したい部分のサイズに関わらずデータ全体を復元するだけの時間がかかってしまう．そこで，本研究では圧縮後のデータから任意の一部分のみを高速に復元できるような圧縮データ構造を提案する．LZ78やLZWなどの圧縮アルゴリズムを拡張し，アルゴリズム本来の圧縮データに加えて，部分復元に必要な最小限の情報も保存することで，このデータ構造を実現した．

本文の構成は以下の通りである．まず，2節において提案手法の元にしたアルゴリズムを述べる．次に3節では2節のアルゴリズムを部分復元用に拡張したアルゴリズムを提案する．4節で提案手法と他アルゴリズムとの比較実験結果を示し，最後に5節でまとめを述べる．

1.1 関連研究

関連研究として [6] が挙げられる．[6] では，LZ78アルゴリズムにより生成された辞書木をいくつかの情報に分けて保存することで，word RAMモデルにおいて，$O(\log_\sigma n)$ 文字の部分文字列を定数時間で復元できるデータ構造を提案している．本研究では，[6] よりも圧縮率および，実装の容易さを重視したデータ構造を提案する．

2 準備

本研究では，LZ78アルゴリズムおよびそれを元に考案されたLZWアルゴリズムを用いた．この2つのアルゴリズムは，入力文字列をフレーズと呼ばれる部分文字列に増分分解していくというもので，入力文字列が長くなるにつれて平均符号長がエントロピーに収束するという特徴を持っている．さらに，実装も容易であることからこれらを元にして，それぞれのアルゴリズムの出力符号列と，それとは別に部分復元に必要な情報を保存することで，圧縮データからの部分復元を行うことができるような手法を考案した．以下では提案手法の元となった2つのアルゴリズムと，部分復元に必要な情報を保存するための索引について述べる．

2.1 LZ78アルゴリズム [8]

LZ78アルゴリズムは，1978年にJ.ZivとA.Lempelが発表したアルゴリズムである．LZ78では辞書を作りながら符号化・復号化を行う．なお，実際はこの辞書に対応する木を構築することで，入力文字列を読み込みながら，既に辞書で入っているフレーズと呼ばれる部分文字列の中から最長一致するものを探し，最長一致したフレーズの末尾に1文字加えた文字列を新たにフレーズとして辞書に登録する操作を繰り返している．以下にLZ78のアルゴリズムを記す．なお，アルゴリズムは [9] を参考にした．

2.1.1 符号化アルゴリズム

1. 初期設定

   辞書に対応する木として，根だけを持つ木を用意する．そして，現在符号化を行っている文字の位置を示す変数を i を $i \leftarrow 1$，現在の出力符号が出力符号列中で何番目であるかを示す変数 j を $j \leftarrow 1$ とする．

2. j 番目の符号語の構成

   i 番目の文字から始まる文字列と最長一致するフレーズを辞書から探す．最長一致したフレーズの番号を R_j とし，文字列中の次の 1 文字を x_j とするとき，(R_j, x_j) を出力する．なお，入力文字列の最後まで達したときは，R_j のみを出力して符号化を終了する．次に，最長一致したフレーズの長さを l としたとき，符号化位置を表す変数 i を $i \leftarrow i+l+1$ と更新する．

3. 辞書の更新

   新たなノードをつくり，その番号を j とする．このノード j とノード R_j とを，ラベル x_j を付けた枝で結ぶ．この時点で辞書には，j 番目の出力符号に対応するフレーズを登録したことになる．次に出力符号の番号を表す変数 j を $j \leftarrow j+1$ として，2. へ戻る．

2.1.2 復号化アルゴリズム

1. 初期設定

   辞書に対応する木として，根だけを持つ木を用意する．そして，現在の出力符号が出力符号列中で何番目であるかを示す変数 j を $j \leftarrow 1$ とする．

2. 出力符号の復号

   j 番目の出力符号から R_j と x_j を求めた後，ノード番号 R_j に対応するフレーズを辞書から探す．そして，そのフレーズの末尾に x_j を加えたものが，この出力符号に対応するフレーズとなる．なお，出力符号として，R_j しかない場合，すなわち最後の出力符号の場合は，番号 R_j に対応するフレーズを辞書から探して出力し，復号を終了する．

3. 辞書の更新

   新たなノードをつくり，その番号を j とする．このノード j とノード R_j とを，ラベル x_j を付けた枝で結ぶ．この時点で辞書には，j 番目の出力符号に対応するフレーズを登録したことになる．次に，出力符号の番号を表す変数 j を $j \leftarrow j+1$ として，2. へ戻る．

図 1: T に対する LZ78 の出力符号列

No	Phrase
0	
1	a
2	aa
3	b
4	bb
5	aaa
6	ab

図 2: T に対する LZ78 の辞書

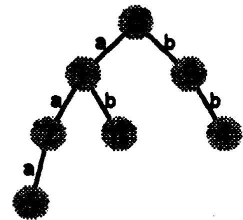

図 3: T に対する LZ78 の辞書に対応する木

T =aaabbbaaaab として，これを LZ78 で圧縮した場合の出力符号列が図 1 であり，この場合の出力符号列は "0a 1a 0b 3b 2a 1b" となる．また，辞書とその辞書に対応する木を図 2 と図 3 で示している．

2.2 LZW アルゴリズム [7]

LZW アルゴリズムは LZ78 を元に 1984 年に T. A. Welch が開発したアルゴリズムである．基本は LZ78 と同じだが，最初から辞書に 1 文字からなるフレーズをすべて登録しておくことで，符号化の際に LZ78 ではフレーズ番号とアルファベットの組であった出力符号がフレーズ番号のみとなっている．

LZW についての詳しいアルゴリズムは本稿では省略するが，アルファベットサイズが $\sigma = 256$，すなわちあらかじめ 1 文字のフレーズ 256 個を辞書に登録している場合，先ほどと同じ T =aaabbbaaaab に対して

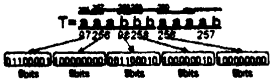

図 4: T に対する LZW の出力符号列

No	Phrase
0	……
……	……
97	aa
98	b
……	……
255	……
256	aa
257	aab
258	bb
259	bba
260	aaa

図 5: T に対する LZW の辞書

LZW を適用した例が図 4 であり，この場合，“97 256 98 258 256 257” という出力符号列が得られる．また，このときの辞書は図 5 のようになっている．最初の時点で辞書にはフレーズ番号が 0 から 255 までの 1 文字のフレーズが入っており，“a” と “b” は ASCII コード上では 97 と 98 に対応しているので，そのフレーズ番号もそれぞれ 97 と 98 である．符号化を開始し，先頭から T を読み込んだとき，最長一致するフレーズはフレーズ番号 97 の “a” である．そこで，97 を出力し，1 文字目の “a” に 2 文字目の “b” を加えた “ab” を 256 番目のフレーズとして辞書に登録する．そして，次は先ほど一致しなかった文字，すなわち 2 文字目の “b” から同じ操作を繰り返す．すると，今度はフレーズ番号 “b” に最長一致し，その次の文字が “a” なので，98 を出力し，“ba” を 257 番目のフレーズとして辞書に登録する．これを繰り返すことで，出力をフレーズ番号のみとした符号化を行うことができる．

2.3 索引 [3], [4]

部分復元に必要なデータを保存するためのデータ構造として，1 と 0 からなるビット列 $B[1..n]$ に対して，以下の 2 つの操作を定数時間で行うための索引を作る．

- $rank(i, B)$
  $B[0..i]$ に含まれる 1 の数を返す．
- $select(i, B)$
  B 中で先頭から i 番目の 1 の位置を返す．

B を元に索引を作り，それを保存しておくことで，B 自体は保存することなく 2 つの操作を定数時間で行うことができる．なお，索引のサイズは $o(n)$ となり，B に含まれる 1 の数が少ないほど小さくなるという特徴がある．

3 提案手法

まず，本研究で扱う部分復元問題を定義する．
問題:入力文字列 $T[1..n]$ を圧縮したデータから，その部分文字列である k 文字 ($T[i..i+k-1]$) を復元する．

本研究では，LZ78 および LZW を用いて，それらの出力符号列と別に，部分的な復元を行うために必要なデータを索引として保存する．提案手法の詳細は以下の通りである．

3.1 LZ78 部分復元アルゴリズム

3.1.1 符号化

符号化アルゴリズムは前述の LZ78 と同じだが，符号化の際に，

$$B[i] = \begin{cases} 1 & T[i] \text{ がフレーズの先頭文字} \\ 0 & \text{それ以外} \end{cases}$$

と定義される長さ n の配列 B を作る．符号化終了後，B に対して $rank$ と $select$ の 2 つの操作を定数時間で行うための索引を作成し，それを LZ78 の出力符号列と一緒に保存して終了する．

3.1.2 復号化

復元は，$T[i]$ が含まれる先頭フレーズと $T[i+k-1]$ が含まれる最終フレーズ，そしてその間の中間フレーズの 3 段階に分けて考える．本手法では，復元範囲が含まれているフレーズをフレーズ単位で復元する．その際，フレーズの復元はそのフレーズの末尾の文字から先頭の文字に向けて順に行っていく．

1. 初期設定

   まず，復元した文字列を格納する配列 $T'[1..k]$ と，現在復元中のフレーズ番号を表す変数 p，p 番

目のフレーズの末尾l文字を除いたフレーズの番号を表す変数p_lを用意し，これまでに復元した文字数を表す変数$count$を$count \leftarrow 0$とする．また，復元中のフレーズの長さを表す変数をrとする．なお，p番目のフレーズに対応するp番目の出力符号は(R_p, x_p)と表す．$T[i]$および$T[i+k-1]$が含まれているフレーズの番号をそれぞれ，$p_{start} \leftarrow rank(i, B)$，$p_{end} \leftarrow rank(i+k-1, B)$として求める．

2. 先頭フレーズの復元

$p \leftarrow p_{start}$，$p_0 \leftarrow p$とする．先頭フレーズはその末尾から$select(p+1, B) - i$文字を求めるだけでよいので，$r \leftarrow select(p+1, B) - i$とする．$1 \leq l \leq r$の各$l$について，$l = 1$から順に$p_l \leftarrow R_{p_{l-1}}$を求める．$l = r$まで$p_l$を求めたら，$1 \leq j \leq r$に対して，$T'[j] \leftarrow x_{r-j}$とする．最後に，$count \leftarrow select(p+1, B) - i$，$p \leftarrow p+1$とする．

3. 中間フレーズの復元

中間フレーズは，そのフレーズの末尾から先頭まで全文字復元する．まず，$p_0 \leftarrow p$, $l \leftarrow 1$とする．そして，$p_l \leftarrow R_{p_{l-1}}$，$l \leftarrow l+1$と更新する操作を$p_l = 0$となるまで繰り返す．$p_l = 0$となった時点での$l$がこのフレーズの長さを表すので，$r \leftarrow l$として，$1 \leq j \leq r$に対して，$T'[count+j] \leftarrow x_{r-j}$とする．最後に，$count \leftarrow count + r$，$p \leftarrow p+1$として，$p < p_{end}$であれば3.を繰り返す．

4. 最終フレーズの復元

最終フレーズはその先頭から$i + k - select(p_{end}, B)$文字復元するだけで良いので，$r' \leftarrow i + k - select(p_{end}, B)$とするが，本手法ではフレーズの復元はその末尾からでないと行えないので，フレーズの末尾から復元する．まず，$p_0 \leftarrow p$, $l \leftarrow 1$とする．そして，$p_l \leftarrow R_{p_{l-1}}$，$l \leftarrow l+1$と更新する操作を$p_l = 0$となるまで繰り返す．$p_l = 0$となった時点での$l$がこのフレーズの長さを表すので$r \leftarrow l$とするが，復元範囲に含まれていない部分は$T'$には格納しないので，$r - (i + k - select(p_{end}, B)) \leq j \leq r$に対して，$T'[count+j] \leftarrow x_{r-j}$とする．

索引を用いることで$rank$と$select$は定数時間で求めることができるので，1文字復元するのは定数時間でできる．しかし，復元にかかる時間は，最長のフレーズ長をhとすると，$O(k+h)$時間となる．これは，4.において最終フレーズの一部しか復元範囲に入っていない場合でも末尾から復元しなければならず，その処理に$O(h)$時間必要となるからである．

3.2 部分復元用LZWアルゴリズム

3.2.1 符号化

LZ78の部分復元用符号化アルゴリズムと同じく，ビット列Bを作りながら符号化を行い，索引を作る．

3.2.2 復号化

復号処理も基本はLZ78と似ているが，LZWでは出力符号がフレーズ番号のみであるため，出力符号にアルファベットが含まれるLZ78のように簡単には求めることができない．そこで，各フレーズの末尾の文字が次のフレーズの先頭文字にもなっていることを利用する．なお，復号処理においては，復元範囲の先頭文字である$T[i]$がp番目のフレーズの末尾かつ$p+1$番目のフレーズの先頭文字である場合，$T[i]$は$p+1$番目のフレーズに含まれているとみなして，$p+1$番目のフレーズを先頭フレーズとする．

1. 初期設定

まず，復元した文字列を格納する配列$T'[1..k]$と，現在復元中のフレーズ番号を表す変数p，p番目のフレーズの末尾l文字を除いたフレーズの番号を表す変数p_lを用意し，これまでに復元した文字数を表す変数$count$を$count \leftarrow 0$とする．なお，p番目のフレーズに対応するp番目の出力符号はR_pとする．$T[i]$および$T[i+k-1]$が含まれているフレーズの番号をそれぞれ，$p_{start} \leftarrow rank(i, B) + \sigma$，$p_{end} \leftarrow rank(i+k-1, B) + \sigma$として求める．

2. 先頭フレーズの復元

$p \leftarrow p_{start}$，$p_0 \leftarrow p$とする．先頭フレーズはその末尾から$select(p+1-\sigma, B) - i$文字を求めるだけでよいので，$r \leftarrow select(p+1-\sigma, B) - i$と

する．$1 \leq l \leq r$ の各 l について，$l = 1$ から順に $p_l \leftarrow R_{p_{l-1}}$ を求める．$l = r$ まで p_l を求めたら，$1 \leq l \leq r$ の各 l に対して，$p_l \geq \sigma$ ならば，$p_l \leftarrow p_l + 1$ として，$p_l \leftarrow R_{p_l}$ と更新する操作を $p_l < \sigma$ となるまで繰り返す．$p_l < \sigma$ となったら，$x_l \leftarrow p_l$ とする．そして，$1 \leq j \leq r$ に対して，$T'[j] \leftarrow x_{r-j}$ とする．最後に，$count \leftarrow count + r$，$p \leftarrow p + 1$ とする．

3. 中間フレーズの復元

   中間フレーズは，そのフレーズの末尾から先頭まで全文字復元する．まず，$p_0 \leftarrow p$，$l \leftarrow 1$ とする．そして，$p_l \leftarrow R_{p_{l-1}}$，$l \leftarrow l + 1$ と更新する操作を $p_l < \sigma$ となるまで繰り返す．$p_l = 0$ となった時点での l がこのフレーズの長さを表すので，$r \leftarrow l$ とする．次に，$1 \leq l \leq r$ の各 l に対して，$p_l \geq \sigma$ ならば，$p_l \leftarrow p_l + 1$ として，$p_l \leftarrow R_{p_l}$ と更新する操作を $p_l < \sigma$ となるまで繰り返す．$p_l < \sigma$ となったら，$x_l \leftarrow p_l$ とする．そして，$1 \leq j \leq r$ に対して，$T'[count + j] \leftarrow x_{r-j}$ とする．最後に，$count \leftarrow count + r$，$p \leftarrow p + 1$ として，$p < p_{end}$ であれば 3. を繰り返す．

4. 最終フレーズの復元

   最終フレーズはその先頭から $i + k - select(p_{end} - \sigma, B)$ 文字復元するだけで良いので，$r' \leftarrow i + k - select(p_{end} - \sigma, B)$ とするが，本手法ではフレーズの復元はその末尾からでないと行えないので，フレーズの末尾から復元する．まず，$p_0 \leftarrow p$，$l \leftarrow 1$ とする．そして，$p_l \leftarrow R_{p_{l-1}}$，$l \leftarrow l + 1$ と更新する操作を $p_l < \sigma$ となるまで繰り返す．$p_l = 0$ となった時点での l がこのフレーズの長さを表すので $r \leftarrow l$ とするが，復元範囲に含まれていない部分は復元しないので，$r - r' \leq l \leq r$ の各 l に対して，$p_l \geq \sigma$ ならば，$p_l \leftarrow p_l + 1$ として，$p_l \leftarrow R_{p_l}$ と更新する操作を $p_l < \sigma$ となるまで繰り返す．$p_l < \sigma$ となったら，$x_l \leftarrow p_l$ とする．最後に，$r - r' \leq j \leq r$ に対して，$T'[count + j - r + r'] \leftarrow x_{r-j}$ とする．

このアルゴリズムの計算量は $O(kh)$ 時間となる．これは，LZ78 の $O(k + h)$ 時間に比べると遅いが，復元の際に一度復元したフレーズをキャッシュしておくことで，再度そのフレーズを復元する必要が生じた場合に高速化できる．

3.3 復元開始位置の限定

B に含まれる 1 の数が少ないほど索引のサイズが小さくなるので，復元の開始位置 i をある正数 $b \leq n$ の倍数のみに制限する．そして，符号化の際に B の代わりとして $B_1[1..n]$ と $B_2[1..c]$ という 2 つのビット列を作って，その索引を保存する．c は T を符号化した際のフレーズ数を表す．i 番目のフレーズの先頭文字を $T[i']$ として，$j = 0, 1, \ldots, \lfloor \frac{n}{b} \rfloor$ とすると，B_1 と B_2 は以下のように定義される．

$$B_1[i] = \begin{cases} 1 & T[i] \text{ が } T[b \cdot j] \text{ を含むフレーズの先頭文字} \\ 0 & \text{それ以外} \end{cases}$$

$$B_2[i] = \begin{cases} 1 & B_1[i'] = 1 \\ 0 & B_1[i'] = 0 \end{cases}$$

$T[b \cdot j]$ が含まれているフレーズの番号 p は，2 つのビット列の索引を用いて，$p \leftarrow select(rank(j, B_1), B_2)$ とすることで，定数時間で求めることができる．

元の B には c 個の 1 が含まれていたが，B_1 に含まれる 1 の数は $O(\frac{c}{b})$ 個となる．つまり，ブロックサイズ b を大きくするほど B_1 中の 1 の数が削減でき，また，B_2 中の 1 の数は B_1 中の 1 の数と等しいので，b の増加に伴って B_1 と B_2 のサイズは減少する．

3.4 開始時点の辞書収録フレーズ数を増やす

LZW の復元時間を短縮するために 2 文字のフレーズもすべて辞書に登録しておく．こうすることで，復元するためにフレーズを前にたどっていく際に，残りが 2 文字になった時点で瞬時にその 2 文字を求めることができる．また，符号化の際に新たに辞書に登録する最短フレーズ長は，あらかじめ 1 文字のフレーズしか辞書に入っていなければ 2 文字となるが，2 文字のフレーズもすべて入っている場合は 3 文字となる．これにより，T を符号化したときのフレーズ数が減少するので，B に含まれる 1 の数も削減でき，索引のサイズも小さくできる．

4 比較実験

提案手法として，以下の3つのアルゴリズムを実装して実験を行った．

(1) 部分復元LZ78（開始位置限定）
LZ78アルゴリズムに対して，3.3を適用した提案手法．

(2) 部分復元LZW（初期フレーズ追加）
LZWアルゴリズムに対して，3.4を適用した提案手法．

(3) 部分復元LZW（開始位置限定・初期フレーズ追加）
LZWアルゴリズムに対して，3.3と3.4を適用した提案手法．

また，比較するために以下の2つのアルゴリズムでも実験を行った．

(4) ブロック化gzip
入力文字列を一定の長さの部分列に分割し，各ブロックごとにgzip(GNU zip)で圧縮したものを連結したファイルを保存し，復元の際は1ブロック全体を復元する．

(5) Static PPM [5]
部分復元用アルゴリズム．

実験に用いたデータは，以下の3つである．

(a) dblp.xml.50MB(52,428,800バイト)
Pizza&Chili Corpus [2] のXMLファイル

(b) kennedy.xls(1,029,744バイト)
The Canterbury Corpus [1] のExcelファイル

(c) plrabn12.txt(481,861バイト)
The Canterbury Corpus [1] のテキストファイル

今回の実験では，各アルゴリズムの圧縮率と100回繰り返し復元するのに必要な時間を調べた．圧縮率は$\frac{\text{出力データサイズ}}{\text{入力データサイズ}} \times 100$であり，値が小さいほど良い．なお，提案手法では，あらかじめメモリに圧縮データと索引を読み込んでおいた状態で復元時間を計測している．Static PPMについては，今回は同じ条件の元で復元時間を計測することができなかったため，圧縮率のみを示す．

4.1 実験結果

図9, 図10, 図11は(a),(b),(c)に対して各アルゴリズムを適用した場合の圧縮率の比較結果である．なお，x軸の値は(1)と(3)については復元開始位置の間隔を，(4)についてはブロックのサイズを表している．(2)および(5)については，この値に関わらず一定の圧縮率となるため，直線で示している．提案手法の中では(3)が最も高い圧縮率となっており，(a)と(b)については，(4)や(5)よりも良い結果となっている．

図6, 図7, 図8は(a),(b),(c)に対して各アルゴリズムを適用して，32バイトを復元した時の圧縮率と復元時間の比較結果である．なお，(1)と(3)については復元位置の間隔が32バイトの場合の圧縮率を表示している。また，(4)についてはブロックのサイズを32バイトから2倍ずつ大きくし，それぞれのブロックサイズにおける圧縮率と、1ブロックを復元するのに必要な時間を表している．gzipは32バイトからバイトまで変化させ，1ブロック延滞を復元した場合の復元時間となっている.．左下にある点ほど圧縮率と復元時間の両面において優れており，復元手法の3つの中では(1)が最も高速に復元できている．(4)の方がブロックサイズが小さい場合は高速な復元が可能となっているが，圧縮率は悪い．

4.2 考察

3.1と3.2の復元時間の計算量にはhが含まれており，$h = O(\sqrt{n})$である．しかしながら，hの値は(a)のファイルでも高々148であり，それほど大きな値にはならないと思われる．

ブロック化gzipは復元は速いが，ブロックサイズが小さい場合は圧縮率が悪い．それに対して，提案手法(3)は復元開始位置の間隔が小さくても圧縮率が良く，Static PPMと比較したときも，提案手法の方が圧縮率が良い場合がある．

5 まとめ

復元文字数が少ない場合は，他アルゴリズムに対しても提案手法の方が良い効果を出している場合が多い．また，提案手法の特徴として，既存のアルゴリズムの

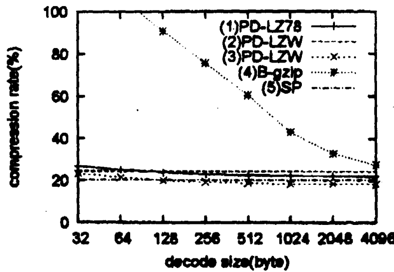

図 6: 圧縮率 (a)

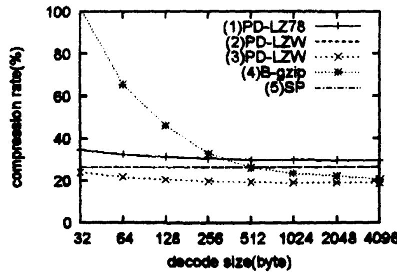

図 7: 圧縮率 (b)

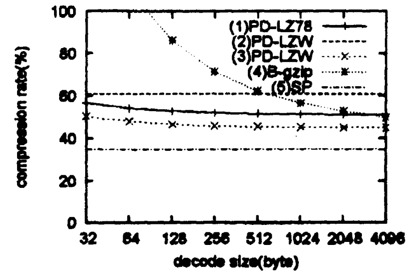

図 8: 圧縮率 (c)

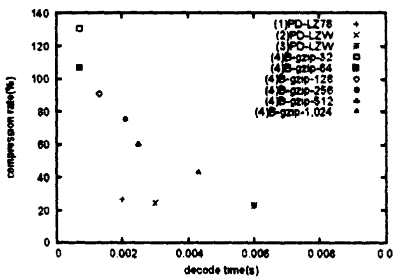

図 9: 圧縮率と復元時間 (a) 復元サイズ 32byte

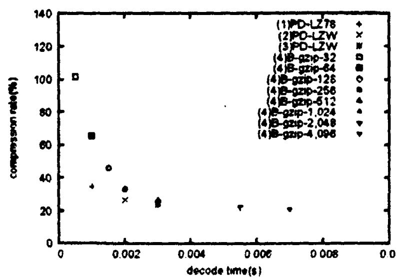

図 10: 圧縮率と復元時間 (b) 復元サイズ 32byte

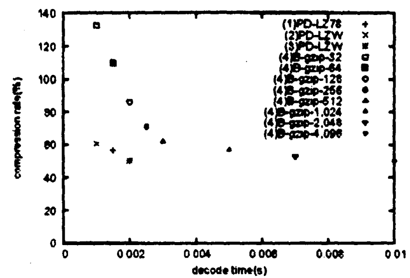

図 11: 圧縮率と復元時間 (c) 復元サイズ 32byte

出力符号列には手を加えていないため，一部分のみを復元したいときは提案手法を使い，全体を復元したいときは元のLZ78もしくはLZWアルゴリズムを用いることで，部分復元と全体復元の使い分けを行うことが可能である.

参考文献

[1] R. Arnold and T. Bell. "A corpus for the evaluation of lossless compression algorithms," In Proc. Data Compression Conference '97 (DCC'97), pp. 201-210, 1997.

[2] P. Ferragina and G. Navarro. "Pizza&Chili Corpus," http://pizzachili.dcc.uchile.cl/.

[3] G. Jacobson. "Space-efficient static trees and graphs," In Proc. IEEE FOCS, pp. 549-554, 1989.

[4] J. I. Munro. "Tables," In Proc. FSTTCS, LNCS 1180, pp. 37-42, 1996.

[5] D. Okanohara. "Partially Decodable Compression with Static PPM," dcc, pp. 471-471, Data Compression Conference (DCC'05), 2005.

[6] K. Sadakane and R. Grossi. "Squeezing Succinct Data Structures into Entropy Bounds," In Proc. ACM-SIAM SODA, pp. 1230-1239, 2006.

[7] T. A. Welch. "A Technique for High Performance Data Compression," IEEE Comput., vol.1, pp. 8-19, 1984.

[8] J. Ziv and A. Lempel. "Compression of indivisual sequences via variable-rate coding," IEEE Trans. Inform. Theory, IT-24(5):530-536, 1978.

[9] 植松友彦. "文書データ圧縮アルゴリズム入門," CQ 出版社, 1994.