電子情報通信学会論文誌 '94/8 Vol. J77-A No. 8

複数の辞書によるユニバーサルデータ圧縮の改良方法について

准員 大峰 恵† 正員 山本 博資††

Universal Data Compression Algorithms with Multiple Dictionaries
Megumu OHMINE†, *Associate Member and*
Hirosuke YAMAMOTO††, *Member*

† 電気通信大学電子情報学科，調布市
Faculty of Electro-Communications, University of Electro-Communications, Chofu-shi, 182 Japan
†† 東京大学工学部計数工学科，東京都
Faculty of Engineering, The University of Tokyo, Tokyo, 113 Japan

あらまし 1次マルコフモデルに基づいて，複数の辞書を使い分けるユニバーサルデータ圧縮アルゴリズムを，LZW符号，Fiala-Greene符号，LZSS符号に適用し，ほぼ10 k～20 kBytesより長いファイルに対して，圧縮率が改善されることを示す．

キーワード ユニバーサルデータ圧縮アルゴリズム，LZW符号，Fiala-Greene符号，LZSS符号

1. まえがき

最近，各種のデータを同一のアルゴリズムで効率良く圧縮できるユニバーサル符号が数多く提案され，その有用性が示されている．それらの中で，実用的に効率の良いユニバーサル符号としては，Lempel-Ziv-Welch (LZW) 符号[1]，Fiala-Greene (FG) 符号[2],[8]，Lempel-Ziv-Storer-Szymanski (LZSS) 符号[3],[4]などがある[5]．

これらのユニバーサル符号では，部分系列 $x_l^{m-1} \triangleq x_l x_{l+1} \cdots x_{m-1}$ を符号化するとき，その時点の辞書を用いて符号化し，次にその x_l^{m-1} を用いて辞書を更新するというように，辞書を適応的に成長させながら符号化が行われる．しかし，このような符号化法では，部分系列 x_l^{m-1} 単位で辞書が更新されるため，x_0^{l-1} と x_l^{m-1} にまたがる系列のつながりやすさは符号化にあまり反映されず，圧縮効率を悪くしている．

この欠点は，x_l^{m-1} を符号化するときに x_{l-1} に基づくマルコフモデルを用いて，複数の辞書を使い分けることにより改善することができる．本論文では，LZW符号，FG符号，LZSS符号に対して，そのような複数の辞書を用いる符号化法を提案する．Plotnik-Weinberger-Ziv (PWZ)[6]は，有限状態機械情報源出力データをその時点の状態 S_{l-1} に基づいて x_l^{m-1} を符号化する方法を提案しているが，本論文で提案する符号化アルゴリズムは，彼らの考えを実用的なユニバーサル符号に適用したものとみなすこともできる．

2., 3., 4. でそれぞれLZW符号，FG符号，LZSS符号に対する改良方式を提案し，それらの性能を5節でシミュレーションにより評価する．なお，主に符号化アルゴリズムを中心に説明するが，復号アルゴリズムは，符号化アルゴリズムの改良に従ってもとの復号アルゴリズムを変更すれば，容易に実現できる．

2. LZW符号

LZW符号は，テーブルを辞書として用いるユニバーサル符号である．x_l^{n-1} を符号化するとき，テーブル内で x_l^{n-1} と最も長く一致する系列 x_l^{m-1} $(m \leqq n)$ を探し，その x_l^{m-1} が格納されているテーブル内の番号を用いて x_l^{m-1} を符号化する．この符号化を x_{l-1} の値に基づいて辞書を使い分けるように改良することを考える．そのために情報シンボル集合 $\mathcal{X}$ に対して，そのシンボル数分の $|\mathcal{X}|$ 個のテーブル $T_\alpha, \alpha \in \mathcal{X}$，を用意し，次のように符号化する．

[複数辞書によるLZW符号アルゴリズム]

(1) (初期化) すべての $T_\alpha, \alpha \in \mathcal{X}$, に対して，$\mathcal{X}$ のすべての要素を登録し，$N_\alpha = |\mathcal{X}| - 1$ とする．$l = 0, x_{-1} = \alpha$ とする．($\alpha \in \mathcal{X}$ は適当に選ぶ．)

(2) x_l^{n-1} と最も長く一致する系列 x_l^{m-1} をテーブル $T_{x_{l-1}}$ 内で検索し，x_l^{m-1} のテーブル $T_{x_{l-1}}$ 内での格納位置番号 M を求める．

(3) CBTcode $(M, N_{x_{l-1}})$ を出力する．

(4) x_l^m を $T_{x_{l-1}}$ に登録する．

(5) $l = m, N_{x_{l-1}} = N_{x_{l-1}} + 1$ とする．

(6) $l = n$ なら終了，$l < n$ なら (2) に戻る．

ここで CBTcode (M, N) は，横尾の Complete-Binary-Tree (CBT) 符号[7]であり，$0 \leqq M \leqq 2^{\lceil \log N \rceil} - N - 1$ なら $\lceil \log N \rceil - 1$ ビットで，また $2^{\lceil \log N \rceil} - N \leqq M \leqq N - 1$ なら，$\lceil \log N \rceil$ ビットで M を表現できる．

3. FG符号

FG符号には，A，B，Cの3種の符号が存在するが，ここでは最も圧縮効率の良い，C符号を考える．

FG符号では辞書として，ウィンドウとパトリシアトリーを用いている．長さ N のウィンドウ内に記憶されている既に符号化されたデータ系列 x_{l-N}^{l-1} に対応してパトリシアトリーが作られ，そのパトリシアトリーを用いて，残りのデータ系列 x_l^{n-1} の符号化が行われる．その後，新しく符号化された x_l^{m-1} をウィンドウにシフトインし，同時に最も古い部分 x_{l-N}^{m-N-1} をシフトアウトし，それに伴いパトリシアトリーを更新す

る．ここでは紙面の都合上，パトリシアトリーの構造や，更新方法などの説明は省略する．詳細は文献(2)を参考して欲しい．

上記のFG符号を1個のウィンドウと$|\mathcal{X}|$個のパトリシアトリー$PT_\alpha, \alpha \in \mathcal{X}$，を用いる，次のような符号化アルゴリズムに改良する．

[複数辞書によるFG符号化アルゴリズム]

（1）（初期化）ウィンドウを空にし，各$\alpha \in \mathcal{X}$に対し，PT_αの根（root）を作る．$x_{-1} = \alpha(\in \mathcal{X})$，$l = 0$とする．

（2）x_l^{n-1}と最も長く一致する系列x_l^{m-1}を$PT_{x_{l-1}}$上で探索し，その情報に基づいてx_l^{m-1}を2値符号化する．

（3）x_l^{m-1}をウィンドウ内にシフトインすると共に，$PT_{x_{l-1}}$にx_l^{m-1}に対応した枝やノードを付加する．また，ウィンドウからシフトアウトしたx_{l-N}^{m-N-1}に対応する枝やノードをすべての$PT_\alpha, \alpha \in \mathcal{X}$，から取り除く．

（4）$l = m$とする．

（5）$l = n$なら終了，$l < n$なら（2）に戻る．

上記（2）におけるx_l^{m-1}の2値符号化部は，

（a）一致系列が存在しない場合，

（b）番号a_Nが割り振られた内部節点に接続された枝の途中まで一致する場合（その枝の長さをb_{Nmax}とし，その枝上の一致する長さをb_Nとする），

（c）番号a_Lが割り振られた葉に接続された枝の途中まで一致する場合（その枝上の一致する長さをb_Lとする），

が存在するが，それぞれの場合に対してもとのFG符号と同じ，Literal, Nodecopy, Leafcopyと呼ばれる2値符号化を用いる．

4. LZSS符号

LZSS符号は，辞書に有限長Nのウィンドウを用いた圧縮方式である．x_l^{n-1}を符号化するとき，ウィンドウ内の系列x_{l-N}^{l-1}から，最も長く一致する系列$x_{l-L}^{m-1-L} = x_l^{m-1}(1 \leqq L \leqq N)$を探し，その一致系列のバッファ内の位置$L$と一致長$m-l$を用いて$x_l^{m-1}$を符号化する．但し，一致長は最大がある$F$に制限されている．Bell[3]は，2分探索木を用いて，この最大一致系列を高速に求めるアルゴリズムを示している．ここでは紙面の都合上，その2分探索木の構成法，探索法，更新法，などは省略する．詳細は文献(3)を参照して欲しい．$x_l^{m-1} = x_{l-L}^{m-L-1}$のとき，$x_{l-1}$の情報を利用すると，$L$より小さい数字で，$x_{l-L}^{m-L-1}$を指定できる．例えば，図1の場合，$x_{14}^{16}$は，$L = 10$シンボル前の$x_4^6$に一致していると表す代わりに，$x_{13} = d$の条件を付けて考え，$d$から始まっている系列だけを考えると，$L_d = 2$個前の系列と一致していると表せる．この考えを取り入れて，LZSS符号化アルゴリズムを1個のウィンドウと，$|\mathcal{X}|$個の2分探索木$BT_\alpha, \alpha \in \mathcal{X}$，を用いた次のようなアルゴリズムに改良する．

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	b	c	d	b	a	a	b	a	d	b	b	a	d	b	a	a	c

図1 改良LZSS符号の符号化
Fig. 1 Encoding of Improved LZSS-code.

[改良LZSS符号アルゴリズム]

（1）（初期化）ウィンドウを空にし，各$\alpha \in \mathcal{X}$に対して，2分探索木BT_αの根（root）を作る．$x_{-1} = \alpha(\in \mathcal{X})$，$l = 0$とする．

（2）$BT_{x_{l-1}}$上で，x_l^{l+F-1}と最も長く一致する系列x_l^{m-1}を探索し，x_{l-1}の条件のもとでのインターバル長$L_{x_{l-1}}$と一致長$m-l$を求め次の符号語を出力する．

（a）一致系列が存在する場合

$$0 \quad + \quad \mathrm{CBTcode}(L_{x_{l-1}} - 1, N_{x_{l-1}}) \quad + \quad \mathrm{CBTcode}(m-l-1, F)$$

（b）一致系列が存在しない場合

$$1 + x_l^{m-1}$$

（3）x_l^{m-1}をウィンドウにシフトインし，x_l^{m-N-1}をウィンドウからシフトアウトする．それに伴い，各BT_αを更新する．

（4）$l = m$とする．

（5）$l = n$なら終了，$l < n$なら2に戻る．

上記(2)(a)の$N_{x_{l-1}}$は，$BT_{x_{l-1}}$に登録されているノード数である．また，一致する系列が存在する場合でも，

$$m - l < (b_{N_{x_{l-1}}} + b_F)/k \qquad (1)$$

を満たす場合は，(b)で符号化する．BT_αには，x_0^{n-1}の符号化手順の途中で現れるαに続く長さFのすべての部分系列を登録する．

LZSS符号は，復号が高速に行える特徴をもっているが，この特性を維持するために，改良LZSS符号の復号は，図2のように同じ種類のシンボルをポインタで結び，L_αと$m-l$から$x_{l-L_\alpha}^{m-L_\alpha-1}$を高速に復号

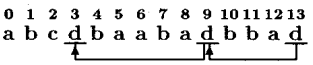

図2 改良 LZSS 符号の復号化
Fig. 2 Decoding of Improved LZSS-code.

できるようにする．例えば，x_{14}^{16} を復号するときは，$L_d = 2$ より，d に関するポインタを2回たどり，実際の位置が求まる．

5. 性能評価

2.～**4.** で説明した LZW 符号，FG 符号，LZSS 符号に対して，それぞれ複数の辞書を用いるように改良した方式の改善効果を図3～5に示す．横軸は，符号化前のファイル長を表し，縦軸は単一の辞書を用いた従来の方式における圧縮率 η と複数の辞書を用いた改良方式における圧縮率 η' との比（すなわち改善率）

$$\lambda = \eta'/\eta \tag{2}$$

である．$\lambda < 1$ ならば改良法の方が圧縮率が優れていることになる．なお，データ系列を構成する情報源シンボルは，8ビット単位で考え，辞書は最大で $2^8 = 256$ 個生成される．FG 符号のウィンドウサイズ N は，$N = 2^{16} = 65536$ とし，LZSS 符号のウィンドウサイズ N および最大一致長 F は，それぞれ $N = 2^{16} = 65536, F = 2^5 = 32$ としている．また，LZW 符号では，簡単のためテーブルサイズは無制限にしている．

各符号に対して，改善率を実行ファイル，英文ファイル，日本語ファイル，C言語のソースファイルに分類して表してある．改善効果はファイルのサイズが大きいほど，またマルコフ性の弱い実行ファイルより英語，日本語，プログラミング言語のようなマルコフ性の強いファイルほど大きい．おおよその目安としては，実行ファイルを除けば，どのユニバーサル符号に対しても，データ長がおおむね 10^4 以上の場合に改善効果がある．

このような特性となる理由は，次の点が考えられる．単一の辞書に比べて複数の辞書を用いると，各辞書が成長するのに時間がかかり，小さなファイルでは十分に辞書が成長する前に圧縮が終わってしまい効率が悪くなる．また，マルコフ性の弱い実行ファイルは，辞書内に長い系列が成長しにくいので同様に効率が悪くなる．特に FG 符号の場合はそのアルゴリズムから，実行ファイルの成長はテキストファイルに比べて，極端に遅く効率が悪い．また，辞書内の系列が1シンボルずつしか成長しない LZW 符号は，他の符号に比べて，小さいファイルに対する圧縮率の悪化が大きい．

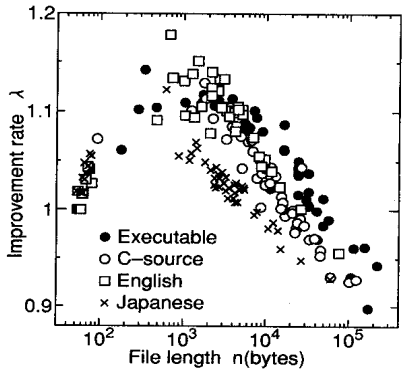

図3 LZW 符号の改善率
Fig. 3 Improvement rate of LZW-code.

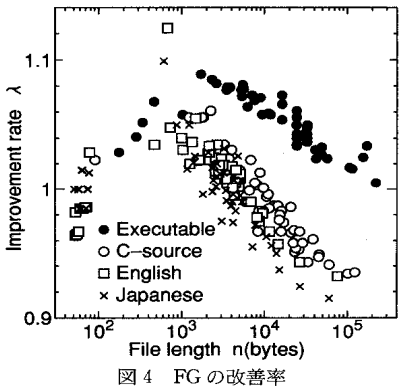

図4 FG の改善率
Fig. 4 Improvement rate of FG-code.

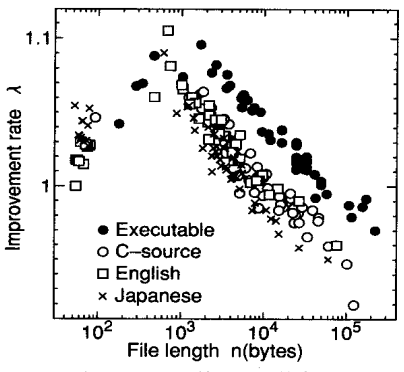

図5 LZSS 符号の改善率
Fig. 5 Improvement rate of LZSS-code.

上記の欠点は，$\alpha \in \mathcal{X}$ に依存する辞書 D_α，と α に依存しない共通辞書 D を用意して，次のように工夫

することにより改善することができると思われる．

• 符号化の初期段階は共通辞書を用いて符号化する．辞書への登録は，$x_{l-1}=\alpha$ に依存して個別辞書 D_α に登録すると共に共通辞書 D にも登録する．

• 個別辞書 D_α が成長し，ある基準を満たせば $x_{l-1}=\alpha$ のとき，D_α を用いて符号化する．さもなければ共通辞書 D を用いる．

6. むすび

本論文では，x_l^{m-1} を符号化するときに x_{l-1} の値に基づいて辞書を使い分ける複数の辞書を用いたユニバーサル符号化法を，LZW符号，FG符号，LZSS符号に適用することにより，ファイルサイズがほぼ10 k～20 kBytes以上の場合，単一辞書を用いるもとの符号より圧縮率が改善されることを明らかにした．今後，ファイルサイズが小さいときに，圧縮率が悪くなる欠点を改善するために，共通辞書を導入した符号化法を検討する必要がある．

文　献

(1) Welch T.A. : "A technique for high performance data compression", IEEE Comp., **17**, 6, pp.8–19 (Jun. 1984).
(2) Fiala E.R. and Greene D.H.: "Data Compression with Finite Windows", Comm. ACM, **32**, 4, pp.490–505 (Jan. 1989).
(3) Bell T.C.: "Better OPM/L Text Compression", IEEE Trans., Common, **34**, pp.1176–1182 (Nov. 1986).
(4) Storer J.A. and Szymanski T.G.: "Data Compression via Textual Substitution", J. ACM, **29**, 4, pp.928–951 (Oct. 1982).
(5) Bell T., Cleary J.G. and Witten I.H.: "Text Compression", Prentice Hall, Inc. (1989).
(6) Plotnik E., Weinberger M.J. and Ziv J.: "Upper Bounds on the Probability of Sequences Emitted by Finite-State Sources and on the Redunduncy of the Lempel-Ziv Algorithm", IEEE Trans., **38**, 1, pp.66–72 (Jan. 1992).
(7) 横尾英俊："ユニバーサル情報源符号化のための修正 Ziv-Lempel 符号"，信学論 (A)，**J68-A**, 7, pp.664–671 (1985–07).
(8) 森田啓義，小林欣吾："制約つき再生可能な文字列分解にもとづく計算機ファイルのデータ圧縮"，情処学論，**33**, 2, pp.110–121 (1992–02).

(平成5年10月26日受付，6年3月1日再受付)