The 26th Symposium on Information Theory
and Its Applications (SITA2003)
Higashiura, Hyogo, Japan, Dec. 15–18, 2003

データ圧縮に適した最頻 digram に基づく逐次型文法変換法
A Method of Sequential Grammar Transform based on the Most Frequent Digram for Data Compression

神田 勝* 石田 崇* 平澤 茂一*
Masaru KANDA Takashi ISHIDA Shigeichi HIRASAWA

Abstract— Recently, the compression algorithms based on the grammar have been proposed. In Sequitur algorithm and the algorithm by E.Yang and J.C.Kieffer, if the sequence has two same digrams, the digram is registered as a rule. While Nakamura et al. and Kanda et al. have proposed algorithms in which the most frequent digram is registered as a rule unsequentially. In this paper, we propose an algorithm in which the most frequent digram is registered as a rule sequentially. And we show advantages of this algorithm by the simulation with Calgary Data. And we show both time and space complexities are $O(N)$, where N is length of the output source sequence.

Keywords— Lossless Compression, grammar-based code, context-free grammars, digram

1 はじめに

確率構造が未知の情報源から出力されるデータ系列に対するデータ圧縮法として，文法に基づく手法が近年盛んに研究されている．文法に基づく圧縮法ではデータ系列を直接符号化するのではなく，データ系列を文法に変換し，それを符号化する．もし，データ系列を効率の良い文法に変換することができれば，元のデータ系列を符号化するよりも文法を符号化することによってより良い圧縮率が達成される．

C.G.Nevill-Manning と I.H.Witten により提案された Sequitur アルゴリズム [1][2] は，繰り返される文字列や単語から階層的な生成規則を逐次的に抽出するアルゴリズムである．この Sequitur アルゴリズムによって階層化された文法と算術符号を組み合わせたデータ圧縮法が提案された．また，E.Yang と J.C.Kieffer は Sequitur アルゴリズムを改良し，圧縮の観点でより効率の良い文法に変換可能なアルゴリズムを提案した（以下 YK アルゴリズム）[3][4]．この手法によって変換された文法に基づく圧縮法は漸近最良性を持つことが証明されている．

これに対して，中村ら，筆者らはデータ系列中の最も頻出する文字列から生成規則を作成する非逐次的な手法を提案している [5][6][7]．この手法は，前述の逐次的な手法よりも圧縮の観点でより効率の良い文法に変換することが可能となる．

以上の手法はいずれも digram と呼ばれる 2 文字（単語）の並びを単位としている手法である．本研究では，最も頻度の高い digram からの生成規則作成を基にし，データ系列を逐次的に文法に変換する手法を提案する．

本手法をカルガリーデータに対するシミュレーションによって評価し，その結果を考察する．また，入力されるデータの長さを N とすると提案手法は計算量及びメモリ量が $O(N)$ であることを示す．

2 準備

2.1 文法

まず初めに文法の基本的な準備を行う．文法は以下のような 4 項組 $G = (\Sigma_T, \Sigma_N, R, S)$ で定義される．

- Σ_T : 終端記号の有限集合．
- Σ_N : 非終端記号の有限集合．
- R : 文の生成規則の有限集合．
- $S(\in \Sigma_N)$: 開始記号．

ただし $\Sigma_T \cap \Sigma_N = \phi$ とする．慣例として終端記号を小文字のアルファベット，非終端記号を大文字のアルファベットで表記することとする．このとき文の生成規則を以下のように定義する．

$$\mu \to \omega \qquad \mu \in \Sigma_N \qquad \omega \in (\Sigma_N \cup \Sigma_T)^*$$

文法に基づく符号では，Σ_T が情報源記号に相当し，データ系列 X は $X \in \Sigma_T^*$ となる．ここで，A^* は集合 A の閉包である．また，$\Sigma = \Sigma_T \cup \Sigma_N$ は記号全体の集合を表し，全順序関係が定義されているとする．

2.2 Sequitur アルゴリズム [1][2]

Sequitur アルゴリズムはデータ系列 X を逐次的に読み込み，digram を単位として，文法に変換する．Sequitur アルゴリズムでは読み込まれた digram が以前に現れている digram と一致した場合，左辺を未使用の非終端記号，右辺を一致した digram とする新しい生成規則を作成し，更に一致した 2 つの digram をこの非終端記号で置き換える．置き換えられた非終端記号も終端記号と同様に 1 記号と見なす．また，生成規則の有限集合 R 内で 1 度しか用いられていない生成規則を削除し，その生成規則を表す非終端記号を，構成している記号列で置き換える．この生成規則を削除し置き換える作業によって，右辺が 3 記号以上となる生成規則が作成される．

以下に Sequitur アルゴリズムの詳細な流れを示す．

＜ Sequitur アルゴリズム＞

1. $S \to \lambda$ [1] という生成規則を作成する．
2. データ系列 X から 1 記号を読み込み，生成規則 S の右辺に追加し現在の記号とする．読み込むべき記号がなければ終了．
3. 現在の記号と直前の記号との digram が R 内に存在するかを調べる．もし存在すれば，その digram が含まれる生成規則の右辺が 3 記号以上のもの，2 記号のもので場合分けを行い，以下の処理を行う．
   - 3 記号以上の場合
     その digram を生成規則の右辺とする新しい生成規則を作り，digram 2 箇所を非終端記号で置き換える．
   - 2 記号の場合
     digram をその生成規則を表す非終端記号で 置き換える．

   存在しなければ 2 へ．
4. もし置き換えられた digram 中に非終端記号が存在し，かつ R 内で 1 箇所にのみ存在するときは，その非終端記号をその生成規則の右辺の記号列で置き換え，その生成規則を削除する．
5. 置き換えた非終端記号も終端記号と同様に扱い，それを現在の記号として 3 へ．

具体的なデータ系列（$X = abcabcbc$）に対する Sequitur アルゴリズムの動作例を表 1 に示す．

* 〒 169-8555 東京都新宿区大久保 3-4-1 早稲田大学理工学部経営システム工学科 Dept. of Industrial and Management Systems Engineering, Schools of Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555 Japan. E-mail: kanda@hirasa.mgmt.waseda.ac.jp

[1] λ は空列を表す．

表 1: Sequitur アルゴリズムの動作例

No.	記号	データ系列	R	備考
1	a	a	$S \to a$	
...				
5	b	$abcab$	$S \to abcab$	ab が一致
			$S \to AcA$	A 作成
			$A \to ab$	
6	c	$abcabc$	$S \to AcAc$	Ac が一致
			$A \to ab$	
			$S \to BB$	B 作成
			$A \to ab$	A が 1 箇所
			$B \to Ac$	
			$S \to BB$	A 削除
			$B \to abc$	
7	b	$abcabcb$	$S \to BBb$	
			$B \to abc$	
8	c	$abcabcbc$	$S \to BBbc$	bc が一致
			$B \to abc$	
			$S \to BBC$	C 作成
			$B \to aC$	
			$C \to bc$	終了

$X = abcabcbc$

2.3 YK アルゴリズム [3][4]

YK アルゴリズムは，Sequitur アルゴリズムと同様，データ系列 X を逐次的に読み込み，digram を単位として生成規則を作成する．しかし，記号を読み込む際に既に作成された生成規則が表す終端記号列とストリングマッチングを行い，完全最長一致した生成規則があれば，それを表す非終端記号を読み込む．これにより，同じ終端記号列を表す生成規則が作成されず，圧縮の観点でより効率の良い文法に変換することが可能になる．

2.4 最頻 digram に基づいた文法変換法 [5][6][7]

Sequitur, YK アルゴリズムはデータ系列から逐次的に生成規則を作成していたが，中村らは最も頻度の高い digram から生成規則を作成する手法を提案した．この手法では，多く出現する digram を生成規則とすることで，頻繁に出現するパターンを生成規則とすることが可能となる．

更に筆者らは [7] において，中村らの手法のように各 digram の頻度が 1 になるまで生成規則を作成することが必ずしも圧縮率を上げることになるとは限らないとし，算術符号の理想符号長 $-\log_2 P(g)$ の計算をしながら[2]，最頻 digram に基づいた文法変換をする手法を提案した．その手法では，理想符号長が最小となる R を採用し，符号化を行う．これらの手法は前述の逐次的な手法より良い圧縮率が達成される．

＜最頻 digram に基づく手法のアルゴリズム [5] ＞

1. データ系列 X を生成規則 S の右辺とする．S の右辺について各 digram の頻度を調べる．
2. 最頻の digram を右辺とする新たな生成規則を作成し，S の右辺内のその digram を非終端記号で置き換える．S の右辺内の各 digram の出現頻度が全て 1 ならば終了．
3. もし置き換えられた digram 中に非終端記号が存在し，かつ R 内で 1 箇所にのみ存在するときは，その非終端記号をその生成規則の右辺の記号列で置き換え，その生成規則を削除する．
4. 2 で置き換えた非終端記号も終端記号と同様に扱い，その直前および直後の記号との digram の頻度を求め 2 へ．

具体的なデータ系列（$X = abcabcbcabc$）に対する最頻 digram に基づく手法のアルゴリズムの動作例を表 2 に示す．

表 2: 最頻 digram に基づく手法の動作例

R	(digram, 頻度)	備考
$S \to abcabcbcabc$	$(ab,3)(bc,4)$ $(ca,2)(cb,1)$	bc が最頻
$S \to aAaAAaA$	$(aA,3)(Aa,2)$	A 作成
$A \to bc$	$(AA,1)$	aA が最頻
$S \to BBAB$	$(BB,1)(BA,1)$	B 作成
$A \to bc, B \to aA$	$(AB,1)$	終了
$X = abcabcbcabc$		

表 3: 提案手法の動作例 ($w = 5$)

No.	R	(digram, 頻度)	備考
5	$S \to [ababc]$	$(ab,2)(ba,1)$ $(bc,1)$	bc が最頻
	$S \to [AAc]$	$(AA,1)(Ac,1)$	A 作成
	$A \to ab$		窓のスライド
6	$S \to AA[ca]$		
7	$S \to AA[cab]$		A と一致
	$S \to AA[cA]$		
	...		
11	$S \to AA[cAcAc]$	$(AA,1)(Ac,3)$ $(cA,2)$	Ac が最頻
	$A \to ab$		
	$S \to A[BBB]$	$(AB,1)(BB,1)$	B 作成
	$A \to ab, B \to Ac$	$(BB,1)$	終了
$X = ababcabcabc$		R 内の [ ] は窓を表す.	

3 提案手法

3.1 従来法の問題点および提案の概要

最頻 digram を基にした手法は，Sequitur, YK アルゴリズムに比べ，圧縮の観点で効率の良い生成規則を作成することができる．しかし，後者が逐次型アルゴリズムであるのに対して，前者は一度全てのデータ系列を読み込んでから処理を行わなくてはならなくなり，遅延が生じてしまう．そこで，本研究ではデータ系列を分割しながら，窓内にデータ系列を読み込み，逐次的に最頻 digram に基づく文法変換をする手法を提案する．

3.2 提案手法のアルゴリズム

1. データ系列 X から，生成規則 S の右辺として窓の大きさ w 分の系列を読み込み，それを窓とする．窓内の各 digram の出現頻度を求める．
2. 窓内の最頻の digram を右辺とする新しい生成規則を作成し，その digram を非終端記号で置き換える．窓内の各 digram の出現頻度が全て 1 ならば，5 へ．
3. もし置き換えられた digram 中に非終端記号が存在し，かつ R 内で 1 箇所にのみ存在するときは，その非終端記号をその生成規則の右辺の記号列で置き換え，その生成規則を削除する．
4. 置き換えた非終端記号も終端記号と同様に扱い，その直前および直後の記号との digram の頻度を求め 2 へ．
5. データ系列から，窓内に記号を 1 記号読み込む（以下「窓のスライド」と呼ぶ）．
6. 直前の記号との digram が R 内に存在するかを調べる．もし存在すれば，その digram が含まれる生成規則の右辺が 3 記号以上のもの，2 記号のもので場合分けを行い，以下の処理を行う．
   - 3 記号以上の場合
     その digram を生成規則の右辺とする新しい生成規則を作り，digram 2 箇所を非終端記号で置き換える．
   - 2 記号の場合
     digram をその生成規則を表す非終端記号で 置き換える．

   5, 6 を繰り返し，w 分の記号の読み込みが終われば 2 へ．
7. 全てのデータ系列に対して，処理を終えたら終了．

具体的なデータ系列（$X = ababcabcabc$）に対する 提案手法のアルゴリズムの動作例を表 3 に示す．

4 計算量及びメモリ量

提案手法が Sequitur アルゴリズム，最頻 digram に基づく手法と同様，計算量及びメモリ量が $O(N)$（N はデータ系列

[2] $P(g)$ は文法 g の経験確率

長）であることを示す．提案手法の計算量を以下の 3 つに分ける．

1. 窓に記号を読み込んだ際に digram の頻度を調べる計算量 (c_1)
2. 最頻の digram を探し，非終端記号で置き換え，頻度情報を更新する計算量 (c_2)
3. 窓スライド時に既に作成された生成規則とのストリングマッチングを行い，非終端記号で置き換える計算量 (c_3)

証明に用いる Notation を以下に示す．

< Notation >

N： 入力データ系列の長さ．

$l_S^{(t)}$： t 時点[3]の生成規則 S の長さ．

$f_d^{(t)}$： t 時点の最頻の digram の頻度．

$|R|$： 最終的に作成された S 以外の生成規則数．

$|\Sigma_T|$： Σ_T の記号数．

[定理] 提案手法の計算量及びメモリ量は $O(N)$ である．

(証明) まず，窓に記号を読み込んだ際，digram の頻度を調べるために窓内の記号列を見て，digram の頻度情報を作成する計算量は N に比例するため，$c_1 = O(N)$ である．

次に，頻度ごとの digram の分類に対する計算量は，最初に digram の頻度を調べた後，各 digram の出現頻度を保持するリストを作成する．この手続きは登録すべき digram の個数だけ計算が必要となる．digram の数は高々 $|\Sigma_T|^2$ であるので，リスト作成の手続きに要する計算量は高々 $O(1)$ である．

また，頻度情報から最頻 digram を探索する計算量は，探索する範囲の digram の数は窓の大きさ w に比例する．w は，$w \ll N$ となるように設定するため，最頻 digram 探索にかかる計算量は $O(N)$ となる．

また，生成規則が新たに作成されたことによるリストの更新のために必要な計算量を考える．この手続きは生成規則の左辺の非終端記号で digram を置き換える際に起こる．このとき 1 回の置き換えによってその digram の前後の 2 つの digram の頻度が減る．一方，置き換えた非終端記号の前後の 2 つの digram が新しく作られる．結果的に 1 つの非終端記号の置き換えに必要な計算量はその digram の頻度 $f_d^{(t)}$ に比例する．従って，最終的に $|R|$ 個の非終端記号について全て置き換える計算量 c_2 は各段階での最頻 digram の頻度の合計

$$F_d = \sum_{t=1}^{|R|} f_d^{(t)}$$

に比例する．ここで，digram を非終端記号へ 1 回置き換えるときに生成規則 S の長さが 1 減ることから，

$$\begin{aligned} l_S^{(1)} &= N - f_d^{(1)} \\ l_S^{(2)} &= N - (f_d^{(1)} + f_d^{(2)}) \\ &\vdots \\ l_S^{(|R|)} &= N - \sum_{t=1}^{|R|} f_d^{(t)} \\ &= N - F_d \end{aligned}$$

が成り立つ．ここで，$l_S^{|R|}$ は最終的な生成規則 S の長さで非負なので，

$$\begin{aligned} l_S^{|R|} &= N - F_d \geq 0 \\ F_d &\leq N \end{aligned}$$

が成り立ち，digram を非終端記号に置き換え，頻度情報を更新する計算量は，$O(N)$ となる．従って，$c_2 = O(N)$ となる．

最後に，窓スライド時のストリングマッチングにかかる計算量も，生成規則内に存在する digram をリストに登録しておくことによって，$O(1)$ の計算量での探索が可能である．従って，$c_3 = O(N)$ となる．

以上より，提案手法に必要な総計算量は $c_1 + c_2 + c_3$ より $O(N)$ である．

一方，メモリ量については，保持しておかなければならない情報は，各生成規則，digram の頻度情報，生成規則内の digram であり，これらは N に比例する．従って，提案手法に必要なメモリ量も $O(N)$ となる． □

[3] t 時点とは t 個目の生成規則を作成した時点を指す．

表 4: 圧縮結果 (設定 1) [bytes]

データ	元サイズ	YK	提案 100	提案 1000
bib	117,543	48,058	45,937	45,532
book1	785,394	349213	337,641	328,447
book2	268,843	115,163	108,769	108,976
geo	102,400	76,240	73,812	73,576
news	387,170	182,235	178,430	177,213
obj1	21,504	13,794	13,548	13,386
obj2	246,814	115,242	111,611	111,032
paper1	54,413	26,902	25,704	25,742
paper2	83,932	40,186	38,293	37,378
paper3	47,628	25,314	24,295	24,031
paper4	15,582	7,982	7,656	7,578
paper5	12,276	7,398	7,059	7,054
paper6	39,126	20,051	19,422	18,805
pic	513,216	79,131	76,730	74,964
progc	41,100	19,887	18,928	19,086
progl	71,908	25,062	24,336	24,182
progp	51,347	17,302	16,723	16,773
trans	90,726	28,762	27,965	28,239

5 シミュレーション結果および考察

本節ではシミュレーションを行い，結果の考察を行う．

5.1 シミュレーション

カルガリーデータ [8] を対象とし，設定 1 では Sequitur アルゴリズムと，設定 2 では YK アルゴリズムと，それぞれ圧縮後のファイルサイズの比較を行う．情報源記号は英数字である．ここで，設定 2 での提案手法は，窓のスライドの際，YK アルゴリズム同様，生成規則が表す終端記号列とのストリングマッチングを行い，完全最長一致した生成規則を表す非終端記号を読み込むこととする．

符号化の前処理で，各手法により変換された文法を一つの系列にすることが必要であり，これを文法の一次元化と呼ぶ．今回は各生成規則に区切りを示す記号を挟むことで一次元化を行い，符号化には算術符号を用いている．それぞれの手法を用いて圧縮した結果をそれぞれ表 4，表 5 に示す．表内の，下付きの数字は窓の大きさを示している．

また，それぞれの手法において作成された生成規則の数 $|R|$，以下の式で示す平均生成規則長 $\overline{L_R}$

$$\overline{L_R} = \frac{\sum_{r_i \in R} (l_{r_i} \times f_{r_i})}{|R|} \quad (1)$$

を調べ，表 6，表 7 に示す．ここで，l_{r_i} は生成規則 r_i が表す終端記号列長，f_{r_i} は生成規則 r_i の R 内の出現頻度を表す．

5.2 シミュレーション結果

1. 表 4，表 5 で，全てのデータにおいて提案手法による圧縮後のファイルサイズは，Sequitur，YK アルゴリズムによるものよりも小さくなり，提案手法の有効性が現れている．
2. 表 6，表 7 で，大部分のデータにおいて，提案手法によって作成された生成規則の数は，Sequitur，YK アルゴリズムによるものよりも少なく，圧縮の観点で効率の良い生成規則の作成が行われていることが分かる．また，大部分の

表 5: 圧縮結果 (設定 2) [bytes]

データ	元サイズ	YK	提案 100	提案 1000
bib	117,543	42,435	39,944	40,608
book1	785,394	321,882	299,446	303,406
book2	268,843	101,870	101,509	101,095
geo	102,400	73,505	72,487	72,686
news	387,170	164,833	160,034	161,742
obj1	21,504	13,185	12,935	12,932
obj2	246,814	107,092	103,389	102,939
paper1	54,413	24,232	23,066	23,987
paper2	83,932	36,403	34,618	34,012
paper3	47,628	23,168	22,030	22,282
paper4	15,582	7,417	7,165	7,246
paper5	12,276	6,892	6,737	6,629
paper6	39,126	18,464	17,412	17,511
pic	513,216	67,501	67,028	65,512
progc	41,100	18,322	17,500	17,726
progl	71,908	22,466	21,235	21,112
progp	51,347	14,944	14,539	14,631
trans	90,726	24,806	24,086	24,265

表 6: 生成規則数 $|R|$ と平均生成規則長 $\overline{L_R}$ (設定 1)

	Sequitur		提案 100		提案 1000	
データ	$\|R\|$	$\overline{L_R}$	$\|R\|$	$\overline{L_R}$	$\|R\|$	$\overline{L_R}$
bib	5,672	28.95	5,064	31.36	5,044	31.66
book1	27,364	35.67	24,201	39.46	23,491	40.54
book2	12,077	29.90	10,628	33.05	10,634	33.08
geo	5,573	18.62	5,033	20.27	4,989	20.42
news	17,707	29.92	16,244	31.93	16,232	32.08
obj1	1,499	17.46	1,407	17.43	1,376	18.10
obj2	12,301	28.36	11,422	30.05	11,369	30.36
paper1	3,469	21.74	3,126	23.56	3,130	23.63
paper2	4,640	24.18	4,165	26.24	4,111	26.68
paper3	3,120	20.47	2,850	21.89	2,845	22.02
paper4	1,125	16.04	1,026	17.23	1,037	17.19
paper5	1,085	15.26	992	16.26	994	16.27
paper6	2,672	20.35	2,466	21.63	2,410	22.20
pic	7,548	99.62	6,502	104.41	6,464	110.05
progc	2,658	21.14	2,395	23.31	2,436	23.06
progl	3,450	31.14	3,192	33.08	3,190	33.80
progp	2,524	30.39	2,313	33.01	2,333	32.89
trans	3,931	35.27	3,639	37.32	3,749	37.01

データにおいて，提案手法によって作成された平均生成規則長の値が，Sequitur，YK アルゴリズムによるものよりも大きくなっていることからも圧縮の観点で効率の良い生成規則の作成が行われていることが分かる．

5.3 考察

1. 提案手法による圧縮後のサイズが，Sequitur，YK アルゴリズムによるものよりも小さくなった．これは，Sequitur，YK アルゴリズムでは，2 つの同じ digram があれば，それを表す生成規則を作成してしまう一方で，提案手法は窓内の最頻 digram から生成規則を作成していくことによる．その結果，表 6，表 7 に示したように，平均生成規則長の値が大きく，無駄な生成規則のない圧縮の観点で効率の良い文法に変換することができたと考えられる．
2. 表 4，表 5 を比較すると，読み込みの際，生成規則が表す終端記号列とのストリングマッチングを行い，完全最長一致したものを表す非終端記号を読み込むことで，圧縮の観点で非常に効率の良い文法に変換できることが分かる．
3. 今回は窓の大きさの設定を 100，1000 としたが，結果に大差はなく，100 程度でも十分に生成規則作成のための情報が得られていると考えられる．

表 7: 生成規則数 $|R|$ と平均生成規則長 $\overline{L_R}$ (設定 2)

	YK		提案 100		提案 1000	
データ	$\|R\|$	$\overline{L_R}$	$\|R\|$	$\overline{L_R}$	$\|R\|$	$\overline{L_R}$
bib	4,619	34.33	4,354	35.85	4,506	34.98
book1	22,773	41.76	21,622	57.43	22,261	42.82
book2	9,814	35.58	9,654	36.29	9,889	35.34
geo	5,031	20.32	4,795	21.19	4,806	21.14
news	14,902	34.90	14,460	35.81	14,698	35.21
obj1	1,345	18.74	1,270	19.29	1,271	19.37
obj2	10,865	31.69	10,286	33.16	10,236	33.32
paper1	2,947	25.03	2,788	26.37	2,934	25.26
paper2	3,945	27.79	3,829	28.59	3,753	29.08
paper3	2,682	23.23	2,635	23.67	2,664	23.47
paper4	1,016	17.58	960	18.48	979	18.11
paper5	963	16.84	908	17.54	907	17.59
paper6	2,359	22.90	2,208	24.16	2,213	24.16
pic	5,409	130.25	5,539	125.03	5,464	127.51
progc	2,336	24.08	2,178	22.51	2,237	24.92
progl	2,945	36.30	2,714	38.85	2,732	38.56
progp	2,027	38.42	1,942	39.91	1,960	39.34
trans	3,212	43.38	3,100	44.71	3,137	44.34

6 まとめと今後の課題

本研究では，最頻 digram を基に，圧縮に適した逐次型文法変換手法を提案し，カルガリーデータに対するシミュレーションにより有効性を示した．また，提案手法の計算量及びメモリ量が Sequitur アルゴリズム，非逐次的な最頻 digram に基づく手法と同様に $O(N)$ であることを示した．

今後は最頻 digram に基づく文法変換法に対して圧縮率の理論的な評価を行いたい．また，本手法で良好な圧縮率を得ることができたので，更に有効な文法変換手法，文法変換から符号化までを逐次的に行えるような手法を提案したい．

7 謝辞

本研究を行うにあたり，数多くのご助言，ご支援を賜りました，早稲田大学 八木秀樹氏，並びに早稲田大学平澤研究室の各氏に感謝いたします．

参考文献

[1] Craig G.Nevill-Manning, Ian H.Witten: "Identifying Hierarchical Structure In Sequences a Linear-Time Algorithm," *Journal of Artificial Intelligence Research* 7, pp.67-82, (1997).

[2] Craig G.Nevill-Manning, Ian H.Witten: "Compression and Explanation using Hierarchical Grammars," *The Computer Journal*, Vol.40, No.2/3, pp.103-116, (1997).

[3] J.C.Kieffer, E.-h.Yang: "Grammar Based Codes: A New Class of Universal Lossless Source Codes," *IEEE Trans. Inform. Theory*, Vol.46, No.3, pp.737-754, (2000).

[4] E.-h.Yang, J.C.Kieffer : "Efficient Universal Lossless Data Compression Algorithms Based on a Greedy Grammar Transform -Part One: Without Context Models," *IEEE Trans. Inform. Theory*, Vol.46, No.3, pp.755-777, (2000).

[5] 中村 博文, 村島 定行: "高頻度隣接の連結に基づいたデータ圧縮法," 第 14 回情報理論とその応用シンポジウム, pp.701-704, (1991).

[6] 中村 博文, 村島 定行: "繰り返し現れる隣接文字の連結に基づくデータ圧縮法," 電子情報通信学会誌, Vol.J79-A, No.7 , pp.1319-1323, (1996).

[7] 神田勝, 石田崇, 小林学, 平澤茂一:"最頻 Digram に基づくデータ圧縮法,"電子情報通信学会技術研究報告, Vol.102,No.198,IT2002-20 ～ 26,pp.31-36,(2002).

[8] ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/