Re-pairアルゴリズムを用いた効率よいVF符号

吉田　諭史†　　喜田　拓也†

† 北海道大学　大学院情報科学研究科　〒 060–0814 北海道札幌市北区北 14 条西 9 丁目
E-mail: †{syoshid,kida}@ist.hokudai.ac.jp

あらまし　本稿では，VF 符号の圧縮率を向上させる方法について論じる．ここで議論する VF 符号とは，分節木と呼ばれる木構造を用いて入力テキストを可変長のブロックに分割し，各ブロックに固定長の符号語を割り当てる情報源符号化法のことである．一方，Re-pair アルゴリズムは，N. Jesper Larsson と Alistair Moffat らによって提案された文法変換に基づく圧縮手法のひとつで，テキスト上の最頻出のバイグラムをテキスト中には出現しない文字で置き換えることを繰り返すことにより圧縮する手法である．本稿では，Re-pair アルゴリズムを用いて，テキスト上の最頻出のバイグラムをテキスト中には出現しない文字で置き換えることを繰り返し，それぞれの文字に固定長の符号語を割り当てる VF 符号を提案する．さらに，本手法では，自然言語のテキストで，典型的な VF 符号である Tunstall 符号に比べて圧縮率が 50%以上向上することと，圧縮が STVF 符号の 2 倍程度高速であることを実験的に示す．
キーワード　歪のないデータ圧縮，VF 符号，文法圧縮

An efficient VF coding using re-pair algorithm

Satoshi YOSHIDA† and Takuya KIDA†

† Graduate School of Information Science and Technology, Hokkaido University
Kita 14-jo Nishi 9-chome, Sapporo 060–0814, Japan
E-mail: †{syoshid,kida}@ist.hokudai.ac.jp

Abstract　In this paper, we discuss how to improve the compression ratio of VF codes. A VF code, we discuss here, is a source coding scheme that parses an input text into variable-length blocks with a dictionary tree, which is called a parse tree, and then encodes them by fixed-length codewords. Re-pair algorithm is a compression method proposed by N. Jesper Larsson and Alistair Moffat. It substitutes the most frequent bigram with a character that does not occur in the text. In this paper, we present a method that uses Re-pair algorithm to improve compression ratio of a VF code. Our algorithm substitutes the most frequent bigram with a character that does not occur in the text to assign a fixed length codeword to each character. In our practice experimental results show that the proposed method improves more than 50% in compression ratio for natural language texts in comparison with the Tunstall code, which is one of the typical VF codes and the compression speed of the proposed method is as twice as that of STVF code.
Key words　lossless data compression, VF code, grammar compression

1. はじめに

テキスト圧縮は，テキスト中に含まれる冗長性をコンパクトに表現することで，記憶のための領域を削減する技術である．これは主に，歪のない情報源符号化を用いて実現される．今日までに，Huffman 符号や Run-length 法，LZ 系圧縮法など，数多くの圧縮手法が提案されており，今なお盛んに研究されている [7,9].

歪のない情報源符号化において，符号とは，情報源アルファベット上の任意の記号列を符号アルファベット上の記号列に写す 1 対 1 写像に他ならない．このような観点からみると，符号化手法は次の 4 種類に大別できる．

FF 符号 (fixed-length-to-fixed-length code)：ある一定の長さ L ごとに情報源系列を分割し，それぞれを固定長 ℓ の符号語へと変換する符号化.

FV 符号 (fixed-length-to-variable-length code)：ある一定の長さ L ごとに情報源系列を分割し，それぞれを可変長の符号語へと変換する符号化.

VF 符号 (variable-length-to-fixed-length code)：可変の長さに情報源系列を分割し，それぞれを固定長 ℓ の符号語へと変換

する符号化.

VV 符号 (variable-length-to-variable-length code): 可変の長さに情報源系列を分割し，それぞれを可変長の符号語へと変換する符号化.

例えば，良く知られた Huffman 符号は，固定長の記号列（1 文字ごと）に可変長の符号語を割り当てる FV 符号である．また，LZ 系圧縮法などは VV 符号とみることができる.

多くの場合において，テキスト圧縮で重要視される要素は圧縮率である．しかしながら，VF 符号は符号語長が固定長であるという制約から，高い圧縮率を達成することが難しい．そのため，現在，可変長符号化である FV 符号や VV 符号の研究が主流である [8]．しかし，近年，テキスト圧縮を利用してパターン照合処理を高速化するという視点から，固定長符号である VF 符号が見直されている [3, 4]．また，符号語長が固定であることから，任意の位置から圧縮テキストを復号することも容易であり，さらに部分的な再圧縮にも都合が良いなど，VF 符号は工学的に多くの利点を持っている．このような背景から，高い圧縮率を達成できる VF 符号の開発が望まれている.

古典的な VF 符号である Tunstall 符号 [12] は，Huffman 符号同様，記憶のない情報源に対してエントロピー符号であることが証明されており，極限では情報源のエントロピーにまで 1 文字あたりの平均符号長が漸近する．しかしながら，実際のところ，その漸近する速度は符号語長に対して非常に緩やかであり，現実的には Huffman 符号ほどの圧縮率を得ることは難しい [20]．これまでに，比較的少数ではあるものの，VF 符号を改善する手法がいくつか提案されている.

Tjalkens と Willems ら [11] は，マルコフ情報源に対する VF 符号の研究に取り組み，実用的な符号・復号化の実装方法を示した．また，Ziv [17] は，マルコフ情報源に対する VF 符号が FV 符号よりも早くエントロピーに漸近することを証明した．1997 年までの FV 符号および VF 符号については，網羅的な調査が Abrahams によってなされている [1]．また，Visweswariah ら [13] は，辞書を用いない VF 符号の性能について報告している．Yamamoto と Yokoo ら [15] は，Tunstall 符号を元に，複数の分節木を切り替えながらテキストを分割する AIVF 符号 (Almost Instantaneous VF code) を提案した．この AIVF 符号は，Huffman 符号並の圧縮率を達成できることが確認されている [16]．また近年，Klein，Shapira ら [4] と Kida [3] は，入力テキストに対する接尾辞木を短く刈り込んだ木を分節木として用いる VF 符号を，各々独自に提案している(注1)．これらは，特に自然言語テキストに対して高い圧縮率を得ることができる [20].

本稿では，Re-pair アルゴリズム [5] を用いて，テキスト上の最頻出のバイグラムをテキスト中には出現しない文字で置き換えることを繰り返し，それぞれの文字に固定長の符号語を割り当てる VF 符号を提案する．また，最急降下法を用いることで，バイグラムの置き換えを途中で打ち切る手法を提案する．さらに，本手法では，英文テキストで，典型的な VF 符号である Tunstall 符号に比べて圧縮率が 30%–40%程度向上することと，最急降下法を用いることで英文テキストの圧縮率を改善できることを示す.

(注1)：前者 [4] は，刈込みを行う際に，接尾辞木を深さ優先探索する DynC (Dynamic Cut) というアルゴリズムを提案しているのに対し，後者 [3] は幅優先的に木のノードを選択していくアルゴリズムを提案している．後者を特に STVF (Suffix Tree based VF) 符号と呼ぶが，本質的には DynC による符号と同じものである.

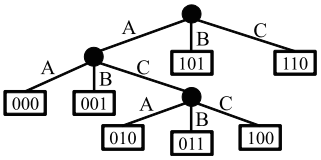

図 1　分節木の例

2. 準　　備

集合 Σ を**有限アルファベット**とする．Σ の要素を**文字**とよぶ．Σ^* は Σ 上の文字列すべてからなる集合である．集合 S の大きさを $|S|$ とかく．よって，アルファベット Σ の大きさは $|\Sigma|$ とかくことができ，これはアルファベットに含まれる文字の種類の数を表す．また，文字列 $x \in \Sigma^*$ の長さを $|x|$ とかく．特に，長さが 0 の文字列を**空語**とよび，ε で表す．したがって，$|\varepsilon| = 0$ である．また，$\Sigma^+ = \Sigma^* - \{\varepsilon\}$ と定義する．二つの文字列 x_1 と x_2 を連結した文字列を $x_1 \cdot x_2$ で表す．特に混乱がない場合は，これを x_1x_2 と略記する．また，$w, x, y, z \in \Sigma^*$ について，$w = xyz$ が成り立つとき，x, y, z をそれぞれ w の**接頭辞**，**部分文字列**，**接尾辞**とよぶ．すべての接頭辞や接尾辞は部分文字列でもあることに注意する．文字列 w の i 番目の文字を $w[i]$ と表し，i 番目から j 番目までの文字を連続して並べた部分文字列を $w[i:j]$ と表す．すなわち，$w[1:j]$ は w の接頭辞であり，$w[i:|w|]$ は接尾辞である．便宜的に，$i > j$ の場合は $w[i:j] = \varepsilon$ と定義する．文字列 w 中の連続する 2 文字をバイグラムとよび，文字の 2 つ組 $(w[i], w[i+1])$ で表す.

木構造のうち，子のあるノードを**内部ノード**，子のないノードを**葉ノード**（または**葉**）とよぶ．親を持たないノード（すなわち木の頂点）を**根**あるいは**ルートノード**とよぶ.

3. VF 符 号

以下に，VF 符号の符号化方法について概説する．先に述べたとおり，VF 符号とは，情報源系列の可変長の部分系列に対して固定長の符号語を割り当てる符号のことである．ただし，Tunstall 符号のように分節木を用いて符号化する方式が最も基本的かつ汎用的であるので，以降では特にこの形式の VF 符号について説明する.

入力テキスト $T \in \Sigma^+$ に対して，これを長さ $\ell \geqq 1$ ビットの符号語で VF 符号化する場合を考える．いま，L 個の葉をもつ分節木 $\mathcal{T}$ が与えられているとする．$\mathcal{T}$ の各葉は，ℓ ビットの整数で番号付けされている．ただし，$L \leqq 2^\ell$ である．このとき，$\mathcal{T}$ によるテキストの符号化は，以下の手順で行われる.

（1） $\mathcal{T}$ の根を探索のスタート地点とする.

（2） 入力テキストから記号を 1 個読み取り，分節木 $\mathcal{T}$ 上

の現節点からその記号でラベル付けされた子へと移る．もし，葉に到達したら，その葉の番号を符号語として出力し，探索の地点を根へ戻す．

（3） ステップ 2 をテキストの終端まで繰り返す．

例えば，図 1 の分節木でテキスト $T := AAABBACB$ を符号化すると，符号語の系列は 000/001/101/011 となる．分節木によって分割された各部分文字列を**ブロック**とよぶ．たとえば，この例において，符号語 011 はブロック ACB を表現している．

VF 符号では，復号時に元の分節木を復元する必要があるため，復元に要する情報を保持しておく必要がある．Tunstall 符号の場合は，Huffman 符号と同様に，文字の頻度表だけから分節木を復元できる．しかしながら，一般には分節木全体を符号化してテキストの符号化系列と共に保存しなくてはならない．一旦，分節木を復元してしまえば，VF 符号の復号は簡単である．符号系列を ℓ ビットごとに区切って読み込み，切り出された符号語に対応する文字列を分節木を参照しながら出力すればよい．

4. 文法圧縮

以下に，文法変換に基づく圧縮について概説する．文法圧縮は，入力された文字列を唯一に生成する文法を作成して，その文法を 2 進数の列に符号化していく圧縮方式である．代表的なものには，LZ 符号 [18, 19] や Byte Pair Encoding [2], Sequitur [6]，Re-pair [5] などがある．特に，LZ 符号は，その圧縮率の高さから多くの変種が存在する [10, 14]．

多くの文法圧縮は，入力された文字列を一意に生成する文脈自由文法を作成する．文脈自由文法は，(Σ, V, σ, R) の 4 つ組で表される．ただし，Σ と V はそれぞれ**終端記号**と**非終端記号**とよばれ，$\Sigma \cap V = \emptyset$ を満たす有限集合である．また，σ は V の要素で，**開始記号**とよばれる．さらに，R は V から $(\Sigma \cup V)^*$ への写像で，**生成規則**とよばれる．

以下で，今回取り扱う Re-pair について簡単に説明する．Re-pair アルゴリズムは，生成規則が

$$\alpha \to \begin{cases} \alpha_1\alpha_2 \dots \alpha_m & \text{if } \alpha = \sigma, \\ \beta\gamma & \text{otherwise} \end{cases}$$

となる文法を構築し，生成規則を符号化する．ただし，すべての $1 \leqq i \leqq m-2$ について，$(\alpha_i, \alpha_{i+1}) \neq (\alpha_{i+1}, \alpha_{i+2})$ を満たす．

以下に，Re-pair アルゴリズムをアルゴリズム 1 に示す．ただし，アルファベット Σ は，0 から $|\Sigma|-1$ までの自然数で表されているものとする．このアルゴリズムでは，現在のテキスト中に最頻出なバイグラムを現在までに出現したことがない記号に置き換えることを繰り返す．ただし，最頻出なバイグラムの出現数が 1 になった場合，記号の置き換えを行わずに，ループから脱出し，符号化する．符号化は，開始記号以外の非終端記号に関する生成規則と開始記号に関する生成規則とに分けて行う．以降，これらをそれぞれ，**辞書部分**と**データ列**とよぶことにする．Re-pair アルゴリズム [5] では，各記号はそれぞれガンマ符号で符号化される．

アルゴリズム 1 Re-pair アルゴリズム

```
1: s ← |Σ|
2: loop
3:    (β, γ) ← 現在のテキスト中に最頻出なバイグラム
4:    f ← テキスト中にバイグラム (β, γ) が出現する回数
5:    if f = 1 then
6:       終了する
7:    end if
8:    辞書に s → βγ を追加する
9:    テキスト中のすべてのバイグラム (β, γ) を s に置き換える
10:   s ← s + 1
11: end loop
12: 辞書とデータ列を符号化する
```

5. 提案手法

本節では，Re-pair アルゴリズムを用いた VF 符号である Re-pair-VF と，最急降下法を用いて計算を途中で打ち切る Truncated Re-pair-VF を提案する．オリジナルの Re-pair アルゴリズムは，辞書部分とデータ列の各記号をガンマ符号を用いて符号化するが，Re-pair-VF では，開始記号を除く記号の数を s とすると，各記号をそれぞれ $\log s$ ビットの符号語に変換して辞書部分とデータ列をそれぞれ符号化する．辞書部分は，非終端記号に順序を付け，生成規則の右辺にある記号のペアを順番に記録していくことで保存する．また，データ列も同様に，開始記号に関する生成規則の右辺にある記号の列をそれぞれ $\log s$ ビットの符号に変換して保存する．

ここで，最急降下法を用い計算を途中で打ち切る手法である Truncated Re-pair-VF を説明する．この手法は，記号の置換によって，文法部分の記述長と圧縮データの記述長の合計の変化を計算する．そして，記述長の合計が長くなる場合に，そこで計算を終了する．以下で，圧縮データの記述長に関して考察する．開始記号を除く記号の数を s とすると，各記号を表すのに $\lg s$ ビット必要である．また，開始記号を除く非終端記号は，$s - |\Sigma|$ 種類あるので，辞書部分の保存には $2(s-|\Sigma|)\lg s$ ビット必要である．この他にデータ列を保存する必要がある．これも，生成規則の右辺にある記号列を記録して保存する．したがって，データ列の保存には，$m \lg s$ ビット必要となる．合計すると，圧縮データの記述長は，

$$2(s-|\Sigma|)\lg s + m \lg s \qquad (1)$$

ビットとなる．

次に，最頻出の記号のペアをひとつの記号に置き換えたときの記述長の変動を辞書部分の変動とデータ列の変動に分けて考える．まずは，辞書部分の変動について考える．新しくひとつの記号を追加するため，各記号は，$\lg s$ ビットから $\lg(s+1)$ ビットになる．つまり，現在の時点での辞書部分は $2(s-|\Sigma|)(\lg(s+1) - \lg s)$ ビットだけ増加する．また，辞書に新しく追加する記号の分は，$2\lg(s+1)$ ビットとなる．

次に，データ列の変動について考える．各記号が $\lg s$ ビットから $\lg(s+1)$ ビットになったため，$m(\lg(s+1) - \lg s)$ ビット

アルゴリズム 2 最急降下法による計算の打ち切り

```
1: s ← |Σ|
2: loop
3:    (β,γ) ← 現在のテキスト中に最頻出なバイグラム
4:    f ← テキスト中にバイグラム (β,γ) が出現する数
5:    if f = 1 or 2(s−|Σ|)(lg(s+1)−lg s)+2lg(s+1)+m(lg(s+
      1) − lg s) ≧ f lg(s + 1) then
6:       計算を終了する
7:    end if
8:    辞書に s → βγ を追加する
9:    テキスト中のすべてのバイグラム (β,γ) を s に置き換える
10:   s ← s + 1
11: end loop
12: 辞書とデータ列を符号化する
```

だけ増加する．しかし，バイグラムをひとつの記号で置き換えるため，最頻出のバイグラムの出現数を f とすると，$f\lg(s+1)$ ビットだけ減少することがわかる．つまり，最頻出のバイグラムをひとつの記号で置き換えたときの記述長の増加量は，

$$2(s-|\Sigma|)(\lg(s+1)-\lg s)+2\lg(s+1)+m(\lg(s+1)-\lg s) - f\lg(s+1) \quad (2)$$

ビットとなる．計算の打ち切りは，合計の記述長が増加する場合に行う．つまり，式 (2) が正になった時点で記号の置換を行わずに計算を打ち切る．

最急降下法による計算の打ち切りのアルゴリズムをアルゴリズム 2 に示す．ただし，アルファベット Σ は，0 から $|\Sigma|-1$ までの自然数で表されているものとする．このアルゴリズムでは，5 行目で最頻出のバイグラムをひとつの記号に置き換えた場合の合計の記述長を計算する．そして，合計の記述長が増加しない限り，最頻出のバイグラムをひとつの記号に置き換えることを繰り返す．合計の記述長が増加することがわかった場合には，記号の置換を行わずに，ループから脱出し，各記号に固定長の符号語を割り当てていく．

6. 実　　験

本稿における提案手法を実装し，圧縮率，圧縮時間，伸長時間の比較を行った．プログラムはすべて C++言語で実装し，GNU g++3.4 でコンパイルした．なお，実験環境は，Intel(R) Xeon(R) プロセッサ 3.00GHz デュアルコア，メモリ 12GB，OS は Red Hat Enterprise Linux ES Release 4 である．比較した手法は，Re-pair アルゴリズムを用いた VF 符号である Re-pair-VF と，最急降下法を用いて計算を途中で打ち切る Truncated Re-pair-VF，オリジナルの Re-pair，典型的な VF 符号である Tunstall 符号 [12] と，Tunstall 符号の改良版である STVF 符号 [20]，gzip の 6 種類である．また，Tunstall 符号および STVF 符号の符号語長を 16 ビットとした．さらに，データとして，日本語コーパス J-Texts(注2) より太宰治全集と，

表 1　実験に使用したテキストファイルの詳細

ファイル名	サイズ（バイト）	内容
dazai.utf.txt	7268943	日本語テキストデータ（UTF-8）
DBLP2003	90510236	XML データ
GBHTG119	87173787	DNA データ
Reuters21578	18805335	英語テキストデータ

GenBank(注3) より GBHTG119，DBLP(注4) の XML データから 2003 年の分を抜き出してきた DBLP2003，英文テストデータである Reuters21578(注5) を選択した．実験に使用したデータの詳細は表 1 のとおりである．

圧縮率の比較を図 2 に示す．ここで，圧縮率は，（圧縮後のデータのサイズ）／（圧縮前のデータのサイズ）で計算した．英文テキストでは，Re-pair-VF の圧縮率は，Tunstall 符号や STVF 符号の圧縮率よりよいことがわかった．特に，Tunstall 符号と比較すると，圧縮率が 50%以上よくなることがわかった．しかし，ゲノムデータでは，Re-pair-VF の圧縮率よりも，Tunstall 符号や STVF 符号の圧縮率のほうが良いことがわかった．また，英文テキストでは，Truncated Re-pair-VF の圧縮率が Re-pair-VF の圧縮率よりもよいことがわかった．ただし，ゲノムデータでは，計算を途中で打ち切ることにより，圧縮率が悪化することがわかった．これは，局所最適解で計算を打ち切っているためである．

圧縮時間の比較を図 3 に示す．Re-pair-VF および Re-pair の圧縮時間は STVF 符号の圧縮時間の半分程度であることがわかった．また，ゲノムデータでは，Truncated Re-pair-VF の圧縮時間が Re-pair-VF の圧縮時間に比べて非常に短いことがわかった．これは，Truncated Re-pair-VF が圧縮の早い段階で計算を打ち切るためであると考えられる．

伸長時間の比較を図 4 に示す．英文テキストおよび日本語テキストでは，Re-pair-VF および Truncated Re-pair-VF の伸長時間が STVF 符号の伸長時間よりも短いことわかった．また，XML データにおける Tunstall 符号の伸長時間が他のデータのものよりも長いのは，圧縮データのサイズが他のものよりも大きいためである．

7. おわりに

本稿では，Re-pair アルゴリズムを用いて，テキスト上の最頻出のバイグラムをテキスト中には出現しない文字で置き換えることを繰り返し，それぞれの文字に固定長の符号語を割り当てる VF 符号を提案した．また，最急降下法を用いることで，バイグラムの置き換えを途中で打ち切る手法を提案した．さらに，本手法では，自然言語のテキストで，典型的な VF 符号である Tunstall 符号に比べて圧縮率が 50%以上向上することと，本手法での圧縮速度は STVF 符号の圧縮速度の 2 倍程度になることを実験的に示した．最急降下法を用いることで自然言語

(注2)：http://j-texts.com/

(注3)：http://www.ncbi.nlm.nih.gov/genbank/

(注4)：http://www.informatik.uni-trier.de/ ley/db/

(注5)：http://www.daviddlewis.com/resources/testcollections/reuters21578/

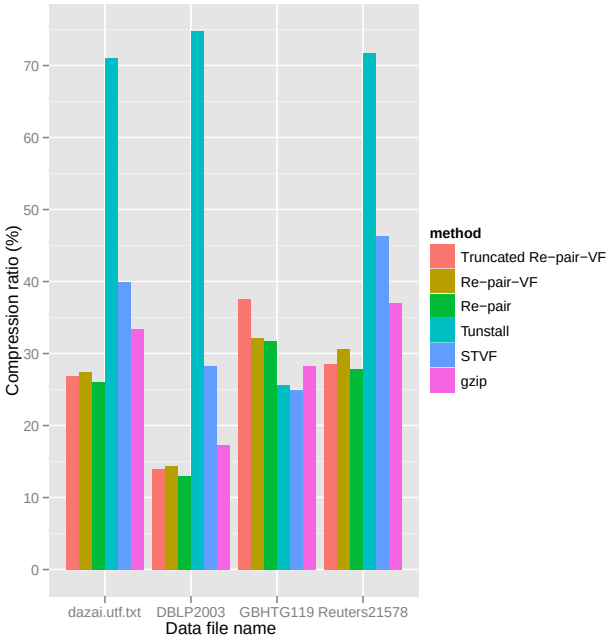

図 2　圧縮率の比較

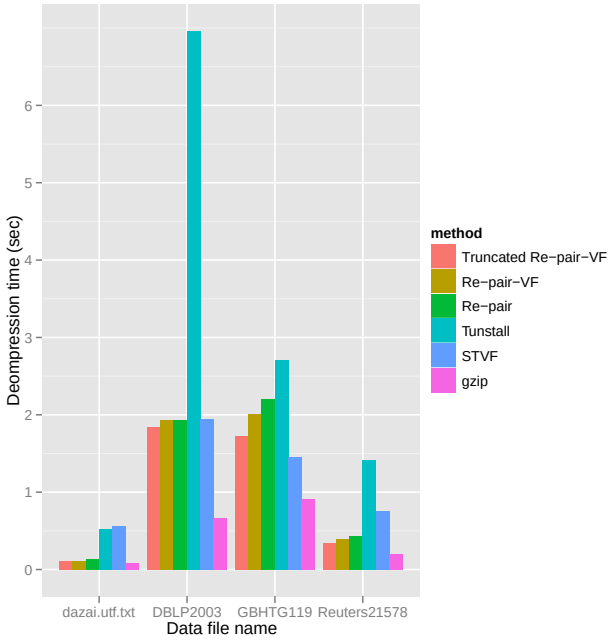

図 4　伸長時間の比較

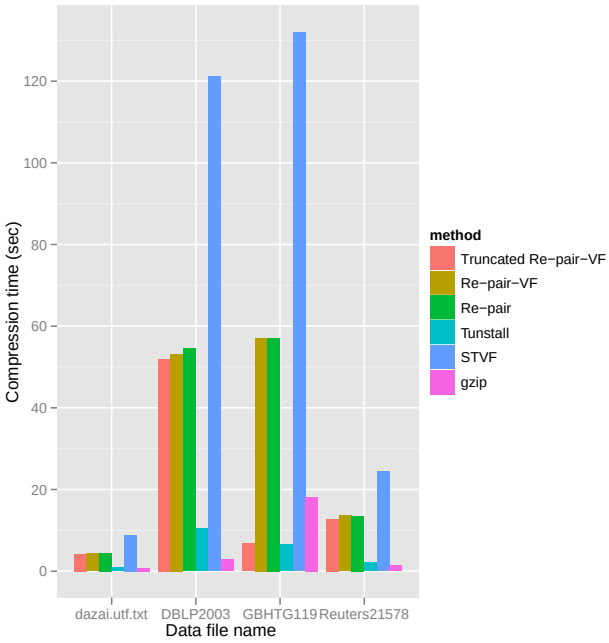

図 3　圧縮時間の比較

のテキストでの圧縮率を改善できることを示した．

謝　　辞

本研究は，文部科学省ブローバル COE プログラム「知の創出を支える次世代 IT 基盤拠点」および科研費（23700002）の助成を受けたものである．

文　　献

[1] J. Abrahams. Code and parse trees for lossless source encoding. In *Compression and Complexity of Sequences 1997*, pages 145–171, June 1997.

[2] Philip Gage. A new algorithm for data compression. *C Users J.*, 12:23–38, February 1994.

[3] T Kida. Suffix tree based VF-coding for compressed pattern matching. In *Proceedings of the Data Compression Conference*, page 449, Mar. 2009.

[4] Shmuel T. Klein and Dana Shapira. Improved variable-to-fixed length codes. In *Proceedings of the 15th International Symposium on String Processing and Information Retrieval*, volume 5280 of *Lecture Notes in Computer Science*, pages 39–50, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Proceedings of the Data Compression Conference 1999 (DCC '99)*, pages 296–305. IEEE Computer Society, 1999.

[6] Craig Nevill-Manning, Ian Witten, and David Maulsby. Compression by induction of hierarchical grammars. In *Proceedings of the Data Compression Conference 1994 (DCC '94)*, pages 244–253. IEEE, 1994.

[7] David Salomon. *Data Compression: The Complete Reference*. Springer, 4th edition, 2006.

[8] David Salomon. *Variable-length Codes for Data Compression*. Springer, Oct. 2007.

[9] Khalid Sayood, editor. *Lossless Compression Handbook*. Academic Press, 2002.

[10] James A. Storer and Thomas G. Szymanski. Data compression via textural substitution. *J. ACM*, 29(4):928–951, 1982.

[11] T. J. Tjalkens and F. M. J. Willems. Variable to fixed-length codes for markov sources. *IEEE Transactions on Information Theory*, IT-33(2):246–257, March 1987.

[12] B. P. Tunstall. *Synthesis of noiseless compression codes*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, 1967.

[13] Karthik Visweswariah, Sanjeev R. Kulkarni, Senior Member, and Sergio Verdu. Universal variable-to-fixed length source codes. *IEEE Transactions on Information Theory*, 47:1461–1472, 2001.

[14] Terry A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.

[15] Hirosuke Yamamoto and Hidetoshi Yokoo. Average-sense

optimality and competitive optimality for almost instantaneous VF codes. *IEEE Transactions on Information Theory*, 47(6):2174–2184, Sep. 2001.

[16] Satoshi Yoshida and Takuya Kida. An efficient algorithm for almost instantaneous VF code using multiplexed parse tree. In *Proceedings of the Data Compression Conference*, pages 219–228. IEEE Computer Society, 2010.

[17] J. Ziv. Variable-to-fixed length codes are better than fixed-to-variable length codes for markov sources. *IEEE Transactions on Information Theory*, 36(4):861–863, July 1990.

[18] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[19] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[20] 喜田 拓也. STVF 符号：頻度刈り込み接尾辞木を用いた効率良い VF 符号化. *DBSJ Journal*, 8(1):125–130, 2009.