大規模テキストに対する共有辞書を用いたRe-Pair圧縮法

関根　渓†　笹川　裕人†　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: †{k_sekine, sasakawa, syoshid, kida}@ist.hokudai.ac.jp

あらまし　1999 年に Larsson と Moffat らによって提案された Re-Pair アルゴリズムは，高い圧縮率を達成する文法圧縮の一つである．しかし，Re-Pair アルゴリズムはオフラインアルゴリズムであり，多くのメモリを使用する．よって，巨大なテキスト上でアルゴリズムを実行するためには，テキスト全体をいくつかのブロックに分割し，ブロック毎に圧縮を行う必要がある．このとき，圧縮時に用いる辞書の一部をブロック間で共有することは，圧縮パフォーマンスの向上に有効であると考えられる．本稿では，テキスト全体からサンプリングしたテキストをもとに，共有辞書を構築した際のパフォーマンスについて論じる．

キーワード　可逆データ圧縮，Re-Pair，文法圧縮，VF 符号

Variable-to-Fixed-Length Encoding for Large Texts Using Re-Pair Algorithm with Efficient Shared Dictionaries

Kei SEKINE†, Hirohito SASAKAWA†, Satoshi YOSHIDA†, and Takuya KIDA†

† Grad. School of Inform. Sci. and Tech., Hokkaido University, N14 W9, Sapporo 060–0814, JAPAN
E-mail: †{k_sekine, sasakawa, syoshid, kida}@ist.hokudai.ac.jp

Abstract　A Re-Pair algorithm proposed by Larsson and Moffat in 1999 is a simple grammar-based compression method, which achieves an extremely high compression ratio. However, since the Re-Pair algorithm is offline and very space consuming, we need to divide a very large text into smaller blocks to apply the algorithm on the very large text. In such case, it is expected that we can improve the compression speed and ratio by sharing a part of the dictionaries among all the blocks. In this paper, we show empirically how the compression speed and ratio vary with constructing the shared dictionary on the text constructed by sampling uniformly from the whole input text.

Key words　Lossless Data Compression, Re-Pair, Grammer-Based Compression, VF Coding

1. はじめに

メモリ装置の性能向上により，数ギガバイトから数十ギガバイト程度のメモリが個人でも容易に手に入れられるようになった．このため，以前は採用が困難であったメモリ消費の激しいアルゴリズムを採用できる場面が増え，データを一旦メモリ上に置いてから処理を行う**オフライン処理**が当たり前のものになっている．データ圧縮技術においても，かつては適応的な処理方式が好ましいとされたが，Burrows-Wheeler 変換 [1] に基づく圧縮法のように，オフライン的な処理をしてでも高い圧縮率を得ることを目的とするアルゴリズムが注目されるようになった．

しかしながら，計算機に搭載されるメモリ容量には限界があり，圧縮プログラムが数百メガバイトを超える巨大なテキストを一度に取り扱うことはやはり困難である．したがって，そのような巨大なテキストを処理する上で，メモリ使用量を節約するような何らかの巧妙な工夫が必要である．例えば，Lemple-Ziv 符号化の派生プログラム [14], [15] の多くは，辞書を作成するときに，参照するテキストや辞書のサイズを制限する工夫がなされている．一方，Bzip2 のように Burrows-Wheeler 変換に基づく圧縮法では，入力テキストを独立した小さいブロックに分割して，ブロック毎に圧縮を行うことで一度に扱うテキストの長さを制限している．

入力テキストをブロックに分割して辞書式圧縮アルゴリズムを用いる場合，全てのブロック間で辞書の一部を共有することで圧縮性能が改善されることが予想される．Wan と Moffat ら [13] は，Larsson と Moffat らによって提案された圧縮法である Re-Pair アルゴリズム [6] を基に，ブロック毎に作成された辞書を全体で統合する手法を提案している．Re-Pair アルゴリズムは文法変換に基づくシンプルなオフライン圧縮アルゴリズムであり，非常に高い圧縮率を達成する．Wan と Moffat らは，Re-Pair の高い圧縮率を保ったままメモリ消費量をうまく

削減できることを実験的に示した [13]．その反面，圧縮速度に大きな犠牲を払っている．

本研究では，より簡単な辞書共有方法を提案し，その圧縮パフォーマンスについて論じる．我々の方法では，あらかじめ準備した共有辞書を全てのブロックで共有する．この方法において圧縮速度と圧縮率は，ブロックサイズ，辞書サイズ，共有辞書サイズ，の 3 つのパラメータに依存する．例えば，共有辞書サイズを大きくすることで，ブロック毎に別々に作られる辞書（**ローカル辞書**）のサイズは小さくなるが，一方で各ブロックの圧縮率は悪化するということが予想される．

本稿では特に，入力テキストを独立したブロックに分割し，Re-Pair アルゴリズムで圧縮する場合に，辞書の共有化が圧縮性能に与える影響について議論する．この共有辞書の構成法として，関根らは，入力テキストの最初のブロックに対して Re-Pair アルゴリズムを適用して辞書を構築し，以降のブロックでは，その一部を共有辞書として継続的に使用する，Blocked-Re-Pair-VF [16] を提案した．この Blocked-Re-Pair-VF では，テキストをブロック化することにより，Re-Pair-VF 従来の圧縮率を殆ど悪化させることなく，メモリの消費を抑えることに成功した．しかし Blocked-Re-Pair-VF では，共有辞書はテキストの最初のブロックから構築されるため，文脈が途中から大きく変わるようなテキストを圧縮する際は，共有辞書の効果が得られなくなる可能性がある．本稿ではその問題を解決するために，入力テキストの部分文字列を繋げ合わせてブロックと同程度のサイズのサンプリングテキストを生成し，そのサンプリングテキストに対して Re-Pair アルゴリズムを適用して辞書を構築し，その辞書を共有辞書としてテキスト全体で使用する，**サンプル法**を提案する．

オリジナルの Re-Pair アルゴリズムでは文法変換後にエントロピー符号化を行うが，今回の手法では辞書のそれぞれのエントリを固定長の符号語で符号化を行う．辞書のあるエントリ（文法の生成規則）はテキスト中の部分文字列に対応している．すなわち，このような符号化方式は，いわゆる **variable-lengh-to-fixed-length 符号（VF 符号）**になっている．VF 符号とは，入力テキストを部分文字列の列に分解し，それぞれの部分文字列に対し固定長の符号を割り当てる符号化方式のことである．VF 符号は，可変長の符号語を用いる圧縮方式と比べて圧縮率の観点から不利であるが，工学的観点から見るといくつか有益な点が存在する．例えば，VF 符号は符号語間の境界が明白なので，圧縮されたテキスト上の任意のブロックに対してランダムに高速なアクセスが可能となる．実際，VF 符号は，圧縮テキストに対するパターン照合の高速化という観点から再評価されている [2], [5]．さらに，副次的な効果ではあるが，VF 符号では共有辞書の効果の分析が容易になるという利点もある．複雑なエントロピー符号化を用いると良い圧縮率を得ることができるが，最終的な出力サイズから共有辞書の効果のみを議論することが煩雑なものとなる．一方，VF 符号は全ての符号語長が等しいため，その効果を圧縮率として直接に観察できる．

今回，我々は提案手法を計算機上で実装し，様々なパラメータの組み合わせで実験を行った．結果としては，実験よって，ほぼ全てのパラメータにおいて，旧手法よりも良い圧縮率を達成することを確認できた．また，適切なパラメータにおいて，bzip2 に匹敵する圧縮率 (27 %程度) を達成することがわかった．

2. 関 連 研 究

これまで，多くの文法圧縮手法が開発されている．LZ78 [15] や，LZW 法 [14]，Bisection [3] は，扱う文法が straight line program のクラスに属する文法圧縮アルゴリズムである．さらに，文法を文脈自由文法（CFG）に限定したアルゴリズムも提案されている [4], [6], [8], [9]．それらのうち，特に Re-Pair [6] や，SEQUITUR [8] は圧縮率に優れている．また，Maruyama ら [7] は，文脈依存文法に基づく圧縮手法を提案している．

VF 符号においては，Klein と Shapira ら [5] や， Kida [2] がそれぞれ独立に，接尾辞木 [12] に基づく VF 符号（STVF 符号）を提案した．STVF 符号では，頻度に基づいて枝刈りを行った接尾辞木を分節木として用いている．STVF 符号は，古典的な VF 符号である **Tunstall 符号** [10] と比べると圧縮率を大幅に向上させたが，gzip などのよく知られた既存ツールには依然およばない．VF 符号の圧縮率を向上するために，Uemura ら [11] は入力テキストを繰り返し読み，分節木をトレーニングする手法を提案している．この手法は，Gzip なみの圧縮率を達成するが，圧縮に非常に時間がかかることが難点である．実際，文献 [11] では，Tunstall 符号の 100 倍程度の時間を要することが示されている．

Wan と Moffat [13] によって提案された Re-Merge アルゴリズムは，大規模テキストに対して Re-Pair アルゴリズムを適用するための拡張手法の一つである，Re-Merge アルゴリズムは，入力テキストを独立なブロックに分割し，各ブロックに対して Re-Pair アルゴリズムを適用して，ブロック毎の辞書を構築する．その後，それらの辞書の統合を段階的に行う．最終的には，統合した辞書を用いて，入力テキスト全体を再度圧縮する．彼らの実験結果によると，Re-Merge アルゴリズムによる圧縮率はきわめて良好であり，英語の自然言語テキストデータである `WSJ508` に対して，およそ 20% の圧縮率を達成している．実験に用いられた `WSJ508` とは，508MB の SGML によるマークアップがなされた新聞記事であり，1987 年から 1992 年までの記事が含まれている．一方で，圧縮時間に関しては芳しくなく，彼らの実験環境（2.8GHz の Intel Xeon，2GB メモリ，Debian GNU/Linux）において 4400 秒かかっている．

3. 圧縮アルゴリズム

本節では，Re-Pair アルゴリズム [6] と，提案手法のサンプル法について説明する．

3.1 Re-Pair アルゴリズム

Re-Pair アルゴリズムはオフラインの文法圧縮アルゴリズムであり，入力としてテキストが与えられると，その入力テキストを一意に生成するような文脈自由文法を生成することでテキストを圧縮する．文脈自由文法とは，句構造文法の一つで，

(Σ, V, σ, R) の四つ組で表される．$\Sigma = \{a_1, a_2, \cdots, a_{|\Sigma|}\}$ は終端記号，$V = \{\alpha_1, \alpha_2, \cdots, \alpha_{|V|}\}$ は非終端記号，$\sigma \in V$ は開始記号である．R は，V から $(\Sigma \cup V)^*$ への関係であり，生成規則とよばれる．以下ではこの生成規則の集合 R を辞書と呼ぶことにする．なお，Σ と V は互いに素な集合である．

Re-Pair アルゴリズムによって，構築される文脈自由文法は以下のようなルールから構成される：

$$\sigma \Rightarrow \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_m} \quad (\forall i_k \in \{1, \cdots, |\Sigma| + |V| - 1\}),$$

$$\alpha_i \Rightarrow \begin{cases} a_i & \text{if } 1 \leqq i \leqq |\Sigma|, \\ \alpha_j\alpha_k \quad (1 \leqq j, k < i) & \text{if } i > |\Sigma|. \end{cases}$$

なお，全ての生成規則の右辺にある 2-gram はユニークである．Re-Pair アルゴリズムは，文字列上に出現する最頻の 2-gram を新しい非終端記号に置き換え，置き換えられた 2-gram を生成規則として R に追加する．この手続きを，文字列上の全ての 2-gram がユニークになるまで繰り返す．開始記号 σ は，この繰り返しが終わった後に得られる文字列に対応している．最後に，辞書 R と σ の内容を適当なエントロピー符号化法を用いて符号化する．

3.2 サンプル法

本論文の提案手法であるサンプル法は，テキストを固定長のブロックに分割し，各ブロックにおいて共有辞書を用いた Re-Pair アルゴリズムを行う．また，後段の符号化には VF 符号を用いる．ここで，B, S, L, C をそれぞれ，ブロック長，共有辞書サイズ，符号語長，サンプリングテキストサイズとする．また，入力テキストを T とおき，それは非負の整数 $\{0, \ldots, |\Sigma| - 1\}$ 上で表されているものとする．このアルゴリズムは B, S, L, C, T を入力として受け取り，圧縮テキスト T' および辞書 D を出力する．サンプル法は，ブロック間で共有する辞書（**共有辞書**）の構築（図 1）と，ブロック毎に固有な辞書（**ローカル辞書**）の構築（図 2）の 2 ステップからなる．以下に，これらのステップの詳細を述べる．

アルゴリズムは，最初に入力テキスト T を長さ B のブロックに分割する．T のサイズを N とおくと，入力テキストは $\lceil N/B \rceil$ 個のブロックに分割される．次に，出来上がったブロック列において，各ブロックの先頭からサイズ c の部分文字列を抜き出し，それらを順に繋げ合わせてサンプリングテキストとする．このとき，$c = C/\lceil N/B \rceil$ とする．このサンプリングテキストに対し，辞書のサイズが S となるまで以下を繰り返す：

（1） サンプリングテキストから最頻出の 2-gram (α, β) を見つける，

（2） 2-gram (α, β) を辞書 D に追加する，

（3） サンプリングテキストのすべての 2-gram (α, β) を新しい記号に置き換える．

以上の処理によって共有辞書の構築が完了する．繰り返しの途中でサンプリングテキスト中のすべての 2-gram がユニークになった場合，そのときの共有辞書のサイズが S に達していない場合であっても共有辞書の構築を終了する．

共有辞書の構築後，各ブロックでの処理を行う．まず，共有辞書を用いて，テキストの左から順に 2-gram の置換を行っていく．共有辞書による置換が終わると，ローカル辞書の構築を行う．ローカル辞書の構築は以下の手順で行われる．

（1） ブロック内から最頻出の 2-gram を発見する，

（2） その 2-gram を辞書 D に登録する，

（3） 登録した 2-gram のブロック内の全ての出現を新しい非終端記号で置き換える．

なお，ローカル辞書の構築は，ブロック内の全ての 2-gram がユニークになった時点または，全体の辞書サイズが 2^L になった時点で終了し，辞書と現在のブロック上の文字列を符号化する．符号化終了後，次のブロックの処理に移る．

辞書の符号化は，文字列の符号化とは別に行われる．符号化された共有辞書に続けて，各ブロックに対する符号化されたローカル辞書を出力する．共有辞書とローカル辞書に登録された 2-gram の各文字は，L ビットの符号語で符号化される．なお，辞書の各エントリに対する符号語の長さが L ビットの固定長なので，エントリや記号を区切る特殊な文字は必要ない．文字列上の各文字も同様に，L ビットの符号語により符号化される．

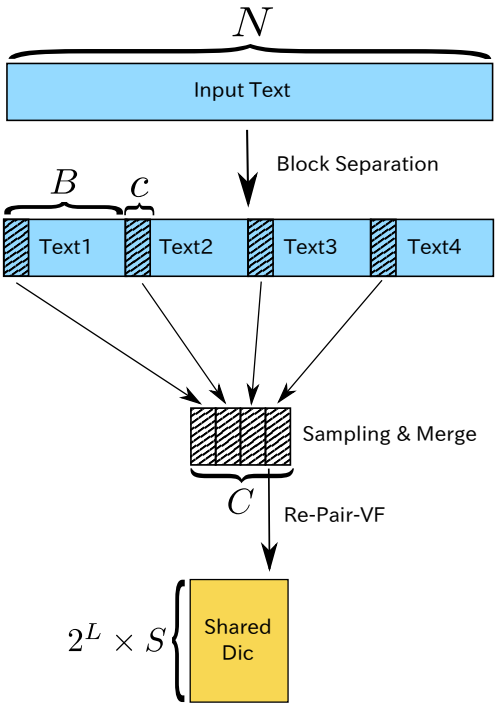

図 1 共有辞書の構築過程．テキストをブロック化した後，各ブロックから一様に部分文字列を抜き出し結合，サンプリングテキストを生成する．そして，サンプリングテキストに対して Re-Pair-VF（Re-Pair でテキストを圧縮した後に，固定長の符号で符号化するアルゴリズム）を適用し，共有辞書を構築する．なお，共有辞書サイズ S は 2^L 個のエントリのうち，共有辞書が占めるエントリ数の割合を表す．

4. 実　　験

提案手法であるサンプル法を計算機上で実装し，実験を行った．なお，実装は C 言語で行い，コンパイラは GCC version 4.6.3 を用いた．

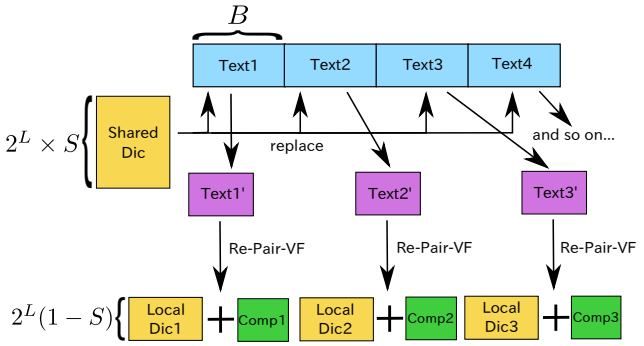

図 2　ローカル辞書の構築およびアルゴリズムの圧縮過程．各ブロックの 2-gram を共有辞書で置き換えた後，個々のブロックに対して Re-Pair-VF を適用し，ローカル辞書と圧縮テキストを生成する．

4.1　実験環境とデータ

実験は，以下の計算機上で行った．

- CPU：Intel Xeon processor 3.6GHz
- メモリ：32GB
- OS：Ubuntu OS 12.04

本実験では，途中で文脈が大きく変わるようなテキストを人為的に作成し入力データとした．具体的には，*Pizza & Chili corpus*(注1) から取得した，DNA のテキストデータ dna，XML のテキストデータ dblp.xml および英文テキストデータ english を順に，サイズがちょうど 2GB になるように繋ぎ合わせた自作テキストデータを生成した．以下の全ての実験はこの自作テキストデータを用いて行った．また，以降は Blocked-Re-Pair-VF を旧手法と呼ぶ．

4.2　旧手法との比較実験

本実験の目的は，文脈が途中で大きく変化するテキストを圧縮する際に，改良された共有辞書が与える効果を実験的に調査することである．実験では，サンプル法に対し，ブロックサイズ B，符号語長 L，共有辞書サイズ S，およびサンプリングテキストサイズ C の 4 つのパラメータを変化させながら，圧縮時間および圧縮率をそれぞれ計測し，旧手法との比較を行った．圧縮時間と圧縮率の結果をそれぞれ，図 3，図 4 にまとめた．図 4 より，圧縮時間に関しては，サンプル法と比べて，旧手法がほぼ全てのパラメータにおいて速い，ということがわかる．旧手法が先頭ブロックから共有辞書を構築するのに対し，サンプル法は共有辞書を構築する前に，そのためのサンプリングテキストを生成するため，余計に時間がかかっていると思われる．図 3 より，圧縮率に関しては，旧手法と比べて，サンプル法がほぼ全てのパラメータの組み合わせで優れていることがわかる．また，サンプル法ではパラメータを変化させても圧縮率の数値が安定しているのに対し，旧手法は共有辞書サイズが大きくなりすぎると圧縮率が極端に悪化している．本実験では，圧縮対象としたテキストデータは文脈が途中で大きく変わるテキストを使用しているため，旧手法では，先頭ブロックと文脈が異なるブロックにおいて，ブロックの文脈に合わない共有辞書でブロックを圧縮する．したがって，圧縮率が著しく悪化している．なお，本実験において最も良い圧縮率は，ブロックサイズ $B = 128$，符号語長 $L = 19$，共有辞書サイズ $S = 1/8$ のとき得られ，その値は 27.78 % であった．

(注1)：*Pizza & Chili corpus*：http://pizzachili.dcc.uchile.cl/texts.html

B (MB)	L (bit)	S	NEW (32M)	NEW (64M)	NEW (128M)	OLD
32	18	1/8	28.77%	29.17%	29.15%	29.33%
32	19	1/8	29.91%	30.29%	30.27%	30.51%
32	20	1/8	31.02%	31.40%	31.37%	31.94%
32	21	1/8	31.99%	32.28%	32.19%	33.43%
32	22	1/8	33.01%	33.10%	32.84%	34.87%
64	18	1/8	28.50%	28.49%	28.49%	28.70%
64	19	1/8	28.88%	28.87%	28.87%	29.08%
64	20	1/8	29.98%	29.96%	29.96%	30.27%
64	21	1/8	31.10%	31.02%	30.99%	31.64%
64	22	1/8	32.18%	31.97%	31.81%	33.06%
128	18	1/8	27.94%	27.95%	27.94%	28.50%
128	19	1/8	27.78%	27.78%	27.78%	28.29%
128	20	1/8	28.24%	28.24%	28.24%	28.76%
128	21	1/8	29.31%	29.28%	29.27%	29.90%
128	22	1/8	30.50%	30.37%	30.26%	31.24%
32	18	3/8	28.50%	28.90%	28.87%	29.71%
32	19	3/8	29.22%	29.55%	29.50%	30.41%
32	20	3/8	30.25%	30.34%	30.20%	31.79%
32	21	3/8	31.51%	31.33%	30.93%	33.21%
32	22	3/8	33.01%	32.64%	32.09%	34.72%
64	18	3/8	28.60%	28.57%	28.56%	29.41%
64	19	3/8	28.59%	28.54%	28.52%	29.37%
64	20	3/8	29.46%	29.29%	29.21%	30.18%
64	21	3/8	30.72%	30.40%	30.01%	31.49%
64	22	3/8	32.18%	31.60%	31.12%	32.91%
128	18	3/8	28.20%	28.20%	28.19%	29.40%
128	19	3/8	27.84%	27.82%	27.80%	28.91%
128	20	3/8	28.16%	27.98%	27.91%	29.03%
128	21	3/8	29.10%	28.91%	28.64%	29.82%
128	22	3/8	30.49%	30.12%	29.72%	31.14%
32	18	5/8	28.57%	28.91%	28.86%	30.87%
32	19	5/8	28.98%	29.11%	29.00%	30.84%
32	20	5/8	30.04%	30.03%	29.68%	31.76%
32	21	5/8	31.51%	31.15%	30.81%	33.16%
32	22	5/8	33.01%	32.64%	31.91%	34.80%
64	18	5/8	28.96%	28.88%	28.85%	30.89%
64	19	5/8	28.78%	28.57%	28.48%	30.34%
64	20	5/8	29.30%	29.17%	28.81%	30.51%
64	21	5/8	30.72%	30.17%	29.87%	31.47%
64	22	5/8	32.18%	31.60%	30.97%	33.50%
128	18	5/8	28.72%	28.66%	28.63%	31.08%
128	19	5/8	28.41%	28.12%	28.04%	30.20%
128	20	5/8	28.17%	28.29%	27.90%	29.84%
128	21	5/8	29.10%	28.77%	28.54%	30.14%
128	22	5/8	30.49%	30.12%	29.66%	31.12%
32	18	7/8	29.45%	29.65%	29.51%	35.23%
32	19	7/8	29.44%	29.36%	29.11%	33.84%
32	20	7/8	30.04%	29.86%	29.48%	33.16%
32	21	7/8	31.51%	31.15%	30.57%	33.42%
32	22	7/8	33.01%	32.64%	31.91%	34.66%
64	18	7/8	30.13%	29.85%	29.71%	35.55%
64	19	7/8	29.82%	29.35%	29.07%	34.03%
64	20	7/8	29.30%	29.58%	29.01%	32.98%
64	21	7/8	30.72%	30.17%	29.71%	32.61%
64	22	7/8	32.18%	31.60%	30.97%	33.48%
128	18	7/8	29.99%	29.70%	29.56%	35.95%
128	19	7/8	29.78%	29.14%	28.85%	34.26%
128	20	7/8	28.17%	29.21%	28.63%	33.03%
128	21	7/8	29.10%	28.77%	28.65%	32.27%
128	22	7/8	30.49%	30.12%	29.66%	32.52%

図 3　サンプル法および旧手法の圧縮率のパラメータによる変化．圧縮率の良い方が緑色，悪い方が赤色となるように色付けを行った．中でも，特に良い値は青字にして下線を引いた．列 B はブロックサイズを，列 L は符号語長を，列 S は共有辞書サイズを，列 NEW (xM) はサンプル法においてサンプリングサイズ $C = x$ (MB) を入力として与えた場合の実験結果を，列 OLD は旧手法の実験結果を表す．なお，共有辞書サイズ S は 2^L 個のエントリのうち，共有辞書が占めるエントリ数の割合を表す．

4.3　サンプル法の挙動の調査

本小節では，ブロックサイズ B と，符号語長 L，共有辞書サイズ S，サンプリングテキストサイズ C の 4 つのパラメータ

B (MB)	L (bit)	S	NEW (32M)	NEW (64M)	NEW (128M)	OLD
32	18	1/8	462.353	469.384	487.365	450.696
32	19	1/8	474.059	482.748	499.019	458.277
32	20	1/8	479.829	489.795	510.272	472.680
32	21	1/8	484.040	493.782	511.064	480.396
32	22	1/8	483.231	494.510	510.823	499.736
64	18	1/8	467.579	473.620	490.592	455.259
64	19	1/8	481.510	489.240	506.401	477.236
64	20	1/8	493.431	502.033	519.011	482.831
64	21	1/8	501.709	513.709	527.536	496.205
64	22	1/8	504.130	511.378	528.882	510.952
128	18	1/8	470.894	478.033	496.123	461.724
128	19	1/8	486.855	494.439	511.045	474.875
128	20	1/8	498.654	507.515	524.142	490.331
128	21	1/8	511.146	519.928	536.475	505.032
128	22	1/8	516.464	525.045	542.233	525.751
32	18	3/8	458.977	465.583	482.757	444.996
32	19	3/8	471.476	481.118	496.744	454.870
32	20	3/8	479.311	489.703	505.657	471.393
32	21	3/8	485.213	496.344	511.866	479.711
32	22	3/8	484.210	496.000	513.527	542.089
64	18	3/8	465.592	472.759	487.364	448.986
64	19	3/8	480.530	487.475	505.623	461.588
64	20	3/8	491.543	500.695	517.394	479.019
64	21	3/8	502.592	510.821	527.270	492.821
64	22	3/8	504.142	512.777	530.693	559.374
128	18	3/8	468.207	475.994	494.279	454.455
128	19	3/8	485.085	494.733	509.170	468.873
128	20	3/8	496.767	505.018	521.957	485.671
128	21	3/8	510.730	516.441	536.732	503.492
128	22	3/8	517.825	525.334	542.770	570.354
32	18	5/8	454.489	466.057	478.684	433.619
32	19	5/8	468.029	476.932	503.813	447.825
32	20	5/8	479.825	487.260	504.423	464.332
32	21	5/8	484.971	496.102	513.263	479.259
32	22	5/8	485.043	498.174	514.887	479.857
64	18	5/8	461.957	468.498	483.922	437.223
64	19	5/8	475.374	483.192	499.820	453.039
64	20	5/8	489.917	493.770	513.247	470.508
64	21	5/8	503.240	509.604	526.293	488.552
64	22	5/8	502.048	515.285	530.325	493.828
128	18	5/8	475.747	473.625	490.701	442.343
128	19	5/8	478.618	488.947	505.329	459.410
128	20	5/8	494.046	500.704	519.131	477.879
128	21	5/8	510.663	515.851	530.917	496.638
128	22	5/8	516.919	520.761	541.908	529.662
32	18	7/8	446.481	455.332	482.253	404.235
32	19	7/8	458.717	474.696	486.557	424.215
32	20	7/8	479.612	479.134	496.204	443.736
32	21	7/8	485.699	496.358	508.588	491.468
32	22	7/8	480.865	492.185	511.150	475.094
64	18	7/8	453.051	462.523	478.626	408.182
64	19	7/8	466.309	475.517	493.851	428.599
64	20	7/8	490.397	486.228	504.920	453.217
64	21	7/8	499.760	510.736	516.128	495.919
64	22	7/8	497.787	509.275	525.899	481.938
128	18	7/8	455.884	466.285	483.428	411.116
128	19	7/8	469.706	481.675	502.244	433.069
128	20	7/8	494.298	491.946	511.525	454.461
128	21	7/8	508.014	517.419	526.538	504.070
128	22	7/8	511.360	518.576	538.300	491.044

図 4　サンプル法および旧手法の圧縮時間（sec）のパラメータによる変化．圧縮率の良い方が緑色，悪い方が赤色となるように色付けを行った．中でも，特に良い値は青字にして下線を引いた．列 B はブロックサイズを，列 L は符号語長を，列 S は共有辞書サイズを，列 NEW (xM) はサンプル法においてサンプリングサイズ $C = x$ (MB) を入力として与えた場合の実験結果を，列 OLD は旧手法の実験結果をそれぞれ表す．

を変化させたときの圧縮時間および圧縮率の変化について考察する．

4.3.1　サンプリングテキストサイズを変化させたときの挙動

サンプル法において，サンプリングテキストサイズ C を変化させながら，圧縮率，圧縮時間を計測した．圧縮時間と圧縮率の結果をそれぞれ，図 5，図 6 にまとめた．

図 5 より，サンプリングテキストサイズを大きくするほど圧縮時間が長くなっていることがわかる．これは，共有辞書の作

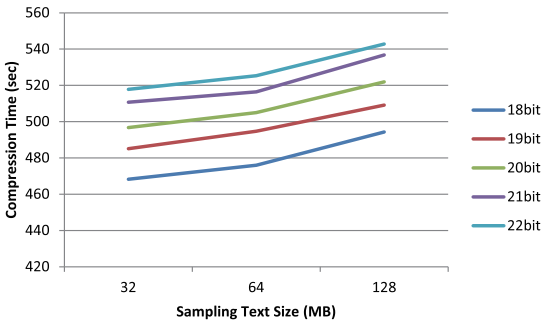

図 5　サンプリングテキストサイズ C (MB) を変化させたときの，各符号語長 L (bit) における圧縮時間の変化．ブロックサイズは $B = 128$ (MB) に，共有辞書サイズは $S = 3/8$ に固定している．

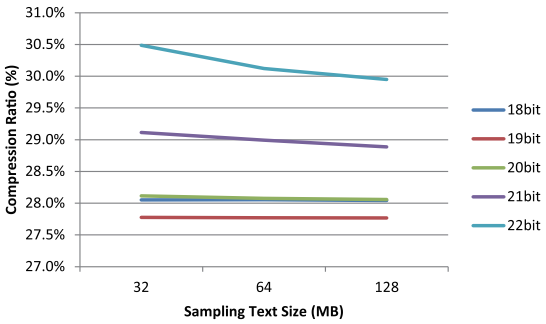

図 6　サンプリングテキストサイズ C(MB) を変化させたときの，各符号語長 L (bit) における圧縮率の変化．ブロックサイズは $B = 128$(MB) に，共有辞書サイズは $S = 3/8$ に固定している．

成時間は，その対象となるテキストサイズに比例するためである．また，図 6 より，圧縮率は殆どの場合において，サンプリングテキストサイズが大きくなるほど，ほんの僅かずつではあるが，改善されていくことが観察できる．これは，共有辞書作成の対象となるサンプリングテキストサイズが増加することによって，より多くの異なる文脈の 2-gram を辞書に登録することが出来るようになり，圧縮率が改善されていると考えられる．また，圧縮率の変化が僅かであるのは，符号語長，共有辞書サイズを固定した状態でサンプリングサイズを変化させているため，辞書に登録できるエントリ数に限りがあり，圧縮率に限界があるためである．

4.3.2　ブロックサイズを変化させたときの挙動

サンプル法においてブロックサイズ B を変化させながら，圧縮率，圧縮時間を計測した．圧縮時間と圧縮率の結果をそれぞれ，図 7，図 8 に示す．

図 7 より，ブロックサイズが大きくなるにしたがって，圧縮時間が大きくなっていることがわかる．これは，ブロックサイズが大きくなるにつれ，ブロック内へのランダムアクセスの増加による，キャッシュミスの発生によって引き起こされる速度低下であると考えられる．図 8 より，圧縮率はブロックサイズが大きくなるにしたがって改善していくことがわかる．出力される圧縮テキストのサイズと辞書のサイズについて調査を行ったところ，圧縮テキストのサイズは，ブロックサイズが小さく

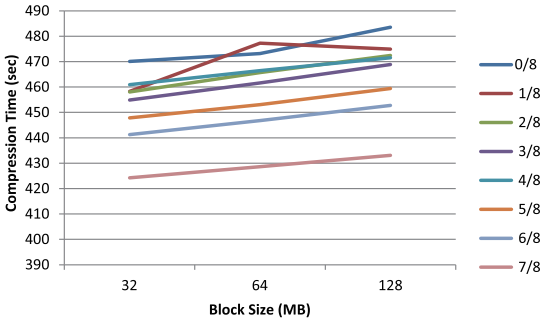

図 7　ブロックサイズ B (MB) を変化させたときの，各共有辞書サイズ S における圧縮時間の変化．サンプリングテキストサイズは $C = 128$ (MB) に，符号語長は $L = 19$ (bit) に固定している．

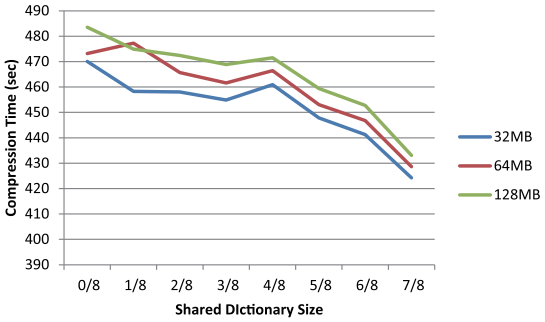

図 9　共有辞書サイズ S を変化させたときの，各ブロックサイズ B (MB) における圧縮時間の変化．サンプリングテキストサイズは $C = 128$ (MB) に，符号語長は $L = 19$ (bit) に固定している．

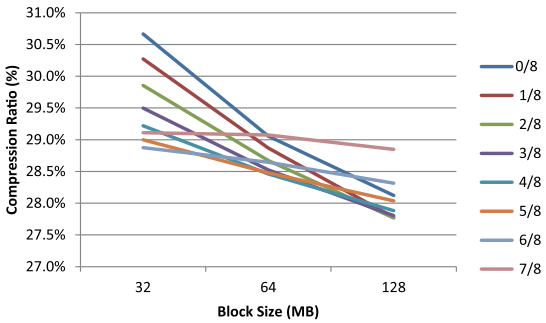

図 8　ブロックサイズ B (MB) を変化させたときの，各共有辞書サイズ S における圧縮率の変化．サンプリングテキストサイズは $C = 128$ (MB) に，符号語長は $L = 19$ (bit) に固定している．

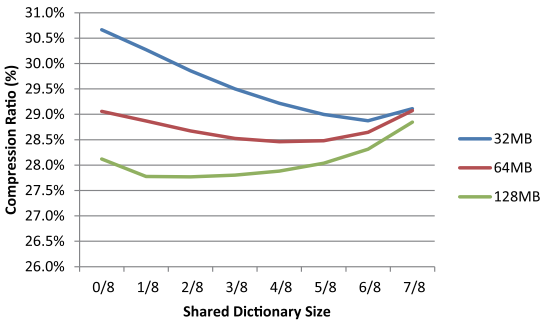

図 10　共有辞書サイズ S を変化させたときの，各ブロックサイズ B における圧縮時間の変化．サンプリングテキストサイズは $C = 128$ (MB) に，符号語長は $L = 19$ (bit) に固定している．

なるほど大きくなるということがわかった．これは，ブロックサイズを小さくするほどテキストのブロック分割が増え，各ブロックに対してよりきめ細やかな圧縮ができるためである．一方，辞書のサイズはブロックサイズが小さくなるほど大きくなるということがわかった．これは，ブロックサイズが小さいと，構築されるローカル辞書の数が増えることで辞書サイズが増加してしまうことに起因する．以上から，ブロックサイズが増大していくとき，圧縮テキストサイズは減少し，辞書サイズは増大していくが，圧縮テキストサイズの減少分の方が大きいため，結果として圧縮率が改善されているということがわかった．

4.3.3　共有辞書サイズを変化させたときの挙動

サンプル法において，共有辞書サイズ S を変化させながら，圧縮率，圧縮時間を計測した．圧縮時間と圧縮率の結果をそれぞれ，図 9，図 10 に示す．

図 9 より，共有辞書サイズが大きくなるにつれて，圧縮時間は短くなっていくことがわかる．これは，共有辞書サイズが大きくなることにより，ローカル辞書サイズが小さくなり，ローカル辞書を作成する時間が減少したためである．図 10 より，圧縮率の変化は，ブロックサイズを固定した場合，共有辞書サイズの変化に関して一様ではなく，ある値で最小となっていることが観察できる．共有辞書サイズの変化による，圧縮テキストサイズと辞書サイズの変化について調査を行ったところ，共有辞書サイズが増加するにつれて，圧縮テキストサイズが大きくなることがわかった．これは，共有辞書サイズが大きくなることで，ローカル辞書サイズが減少し，ブロック特有の 2-gram を辞書のエントリーに格納できないため，圧縮テキストが大きくなる．また，共有辞書サイズが大きくなるにつれ，辞書サイズは小さくなることがわかった．これは，各ブロックで構築されるローカル辞書のサイズが小さくなることで，結果として全体の辞書サイズが小さくなるためである．以上から，圧縮テキストサイズと辞書サイズはトレードオフの関係にあり，さらに圧縮ファイルのサイズは圧縮テキストと辞書サイズの合計で表されるため，圧縮率は共有辞書サイズの変化に対して一様には変化しない．また，共有辞書サイズ S のいくつかのパラメータでは，共有辞書を用いない場合（$S = 0/8$）よりも，圧縮率が良くなることが観察できた．

4.3.4　符号語長を変化させたときの挙動

サンプル法において，符号語長 L を変化させながら，圧縮率，圧縮時間を計測した．圧縮時間と圧縮率の結果をそれぞれ，図 11，図 12 に示す．

図 11 より，符号語長が大きくなるほど圧縮時間が増加することがわかる．符号語長の増加により，辞書に登録できるエントリー数の増加し，辞書構築および出力に時間がかかるためである．また，図 12 より，圧縮率に関しては符号語長の変化に一様ではなく，ある一定の値で最小になっていることがわかる．符号語長が短い場合，辞書のエントリーが少なくなるため圧縮

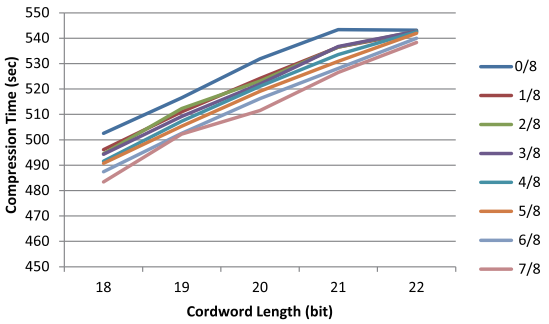

図 11　符号語長 L を変化させたときの，各共有辞書サイズ S における圧縮時間の変化．サンプリングテキストサイズは $C = 128$ (MB) に，ブロックサイズは $B = 128$ (MB) に固定している．

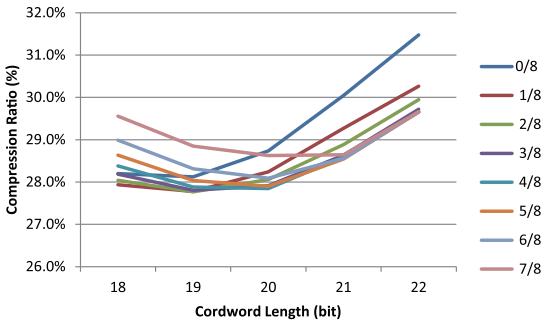

図 12　符号語長 L を変化させたときの，各共有辞書サイズ S における圧縮率の変化．サンプリングテキストサイズは $C = 128$ (MB) に，ブロックサイズは $B = 128$ (MB) に固定している．

テキストのサイズは大きくなってしまうが，逆に出力される辞書のサイズは小さくなる．符号語長が長い場合，辞書のエントリーが多くなるため圧縮テキストサイズは小さくなるが，辞書のエントリーが多くなるため出力される辞書のサイズが大きくなってしまう．このように，圧縮率において，圧縮テキストサイズと出力される辞書のサイズはトレードオフの関係にあるため，両者がうまくバランスがとれるパラメータで，最良の圧縮率を得ることができる．

4.4　サンプル法の圧縮パフォーマンスの評価

サンプル法の圧縮パフォーマンスの一般的な性能を検査するため，上記の自作テキストデータに対して，既存の圧縮手法を適用し，圧縮率，圧縮時間，および伸展時間を計測した．サンプル法との比較に用いたのは，gzip(注2) と，bzip2(注3)，LZMA (7-Zip)(注4)，旧手法の 4 つである．旧手法以外のプログラムはどれも Ubuntu 12.04 に搭載されているパッケージ（gzip 1.4, bzip2 1.0.6，および LZMA 5.1.0 alpha）を使用した．圧縮率，圧縮時間，および伸展時間の結果を図 13，図 14，および図 15 にまとめた．図 13 より，サンプル法は bzip2 に匹敵する圧縮率を達成していることがわかる．また，図 14 より，サンプル法は bzip2 と比べると 2，3 倍程度の圧縮時間がかかっており，

(注2)：zip：http://www.gzip.org/
(注3)：bzip2：http://www.bzip.org/
(注4)：LZMA：http://www.7-zip.org/

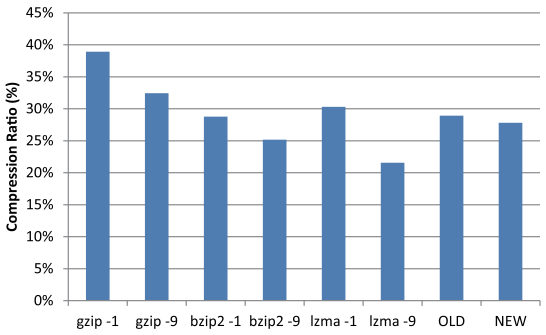

図 13　サンプル法および既存手法の圧縮率．OLD は旧手法を，NEW はサンプル法を表す．

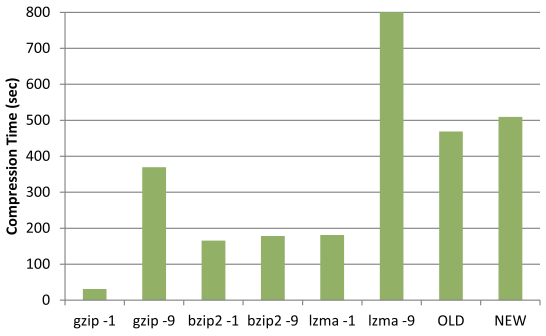

図 14　サンプル法および既存手法の圧縮時間．OLD は旧手法を，NEW はサンプル法を表す．

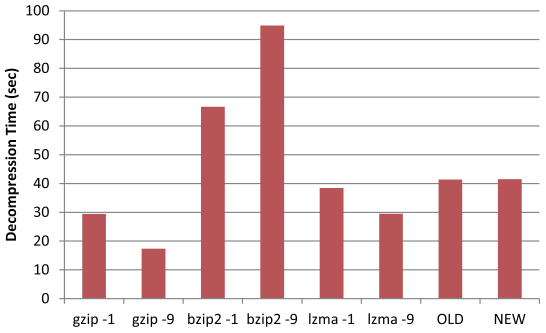

図 15　サンプル法および既存手法の伸展時間．OLD は旧手法を，NEW はサンプル法を表す．

他手法と比べても多くかかっている．一方，図 15 より，伸展時間は圧縮時間の 1/10 程度であり，bzip2 よりも高速である．サンプル法は VF 符号で符号化を行うため，圧縮テキストが非常に扱いやすいという特性がある．したがって，圧縮ファイルを展開せずにパターン照合やデータマイニングなどの処理を行うのであれば，サンプル法は有用となる．

5.　おわりに

本論文では，大規模テキストに対して，VF 符号による圧縮を行うためのアルゴリズムである Blocked-Re-pair-VF に対して，その共有辞書構築法を改良したサンプル法を提案した．提案手法では，途中で文脈が変わるテキストを入力とした場合でも圧縮率を悪化させないように，入力テキストに対して一様に

サンプリングを行って生成したサンプリングテキストから共有辞書を構築するように改良を行った．実験では，ほぼ全てのパラメータにおいて，旧手法よりも良い圧縮率を達成することを確認できた．また，提案手法は固定長の符号語を用いる圧縮手法であるにも関わらず，適切なパラメータにおいて，bzip2 並みの圧縮率を達成することがわかった．

今回，我々の提案手法では，全てのパラメータは，ユーザが入力として与えていた．しかし，現実的には，入力テキストの情報から，適切なパラメータが自動的に決定されることが求められる．

また，今回は，提案手法と Re-Merge アルゴリズム [13] との直接の比較は行っていないが，Re-Merge アルゴリズムは，辞書の統合手法はやや複雑なものであるので，提案手法の辞書を共有する簡単な手法は，圧縮時間において有利であると考えられる．今回の提案手法と Re-Merge アルゴリズムとの直接の比較を行うことも今後の課題である．

文　献

[1] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[2] T. Kida. Suffix tree based VF-coding for compressed pattern matching. In *Proc. of Data Compression Conference 2009 (DCC 2009)*, p. 449, Mar. 2009.

[3] J. C. Kieffer, G. Nelson E.-H. Yang, and P. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Inform. Theory*, Vol. 46, No. 4, pp. 1227–1245, 2000.

[4] J. C. Kieffer and E.-H. Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Trans. on Inform. Theory*, Vol. 46, No. 3, pp. 737–754, 2000.

[5] S. T. Klein and D. Shapira. Improved variable-to-fixed length codes. In *Proc. of the 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008)*, pp. 39–50, 2008.

[6] N. J. Larsson and A. Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, Vol. 88, No. 11, pp. 1722–1732, 2000.

[7] S. Maruyama, Y. Tanaka, H. Sakamoto, and M. Takeda. Context-sensitive grammar transform: Compression and pattern matching. In *Proc. of 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008)*, pp. 27–38, Nov. 2008.

[8] C. Nevill-Manning, I. Witten, and D. Maulsby. Compression by induction of hierarchical grammars. In *Proc. of the Data Compression Conference 1994 (DCC '94)*, pp. 244–253. IEEE, 1994.

[9] H. Sakamoto, T. Kida, and S. Shimozono. A space-saving linear-time algorithm for grammar-based compression. In *String Processing and Information Retrieval*, Vol. 3246 of *Lecture Notes in Computer Science*, pp. 218–229. Springer Berlin / Heidelberg, 2004.

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

[11] T. Uemura, S. Yoshida, T. Kida, T. Asai, and S. Okamoto. Training parse trees for efficient VF coding. In *Proc. of the 17th international conference on String processing and information retrieval (SPIRE 2010)*, pp. 179–184, 2010.

[12] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, Vol. 14, No. 3, pp. 249–260, 1995.

[13] R. Wan and A. Moffat. Block merging for off-line compression. *J. Am. Soc. Inf. Sci. Technol.*, Vol. 58, No. 1, pp. 3–14, January 2007.

[14] T. A. Welch. A technique for high performance data compression. *IEEE Comput.*, Vol. 17, pp. 8–19, June 1984.

[15] J. Ziv and A. Lempel. Compression of individual sequences via variable-length coding. *IEEE Trans. on Inform. Theory*, Vol. 24, No. 5, pp. 530–536, Sep 1978.

[16] 関根渓, 笹川裕人, 吉田論史, 喜田拓也. 共有辞書を用いた効率の良い圧縮アルゴリズム. 第 156 回データベースシステム研究会, 2012 年 12 月.