Vol.2012-DBS-156 No.7
2012/12/12

共有辞書を用いた効率の良い圧縮アルゴリズム

関根 渓[1,a)]　笹川 裕人[1,b)]　吉田 諭史[1,2,c)]　喜田 拓也[1,d)]

概要：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，単純なヒューリスティックに基づいた，非常に高い圧縮率を達成する文法圧縮の一つである．しかしながら，Re-Pair アルゴリズムはオフラインのアルゴリズムであり，また，線形サイズではあるものの多くのメモリを使用する．したがって，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．このような状況において，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，ブロック毎に Re-Pair アルゴリズムを行い，辞書式圧縮の辞書に相当する生成規則の一部をブロック間で共有する手法について論じる．ここでは，抽出された文法の符号化には固定長の符号化を用いている．圧縮パフォーマンスを決定する 3 つのパラメータ（ブロックサイズ，辞書サイズ，辞書における共有辞書の割合）の変化によって，圧縮時間と圧縮率がどのように変化するのかを実験的に示し，その傾向について議論する．

キーワード：文法圧縮，大規模テキスト，ブロック化

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

Kei Sekine[1,a)]　Hirohito Sasakawa[1,b)]　Satoshi Yoshida[1,2,c)]　Takuya Kida[1,d)]

Abstract: The Re-Pair algorithm proposed by Larsson and Moffat in 1999 is a simple grammar-based compression method that achieves an extremely high compression ratio. However, Re-Pair is an offline and very space consuming algorithm. Thus, to apply it to a very large text, we need to divide the text into smaller blocks. Consequently, if we share a part of the dictionary among all blocks, we expect that the compression speed and ratio of the algorithm will improve. In this paper, we implemented our method with exploiting variable-to-fixed-length codes, and empirically show how the compression speed and ratio of the method vary by adjusting three parameters: block size, dictionary size, and size of shared dictionary. Finally, we discuss the tendencies of compression speed and ratio with respect to the three parameters.

Keywords: grammer compression, large text, blocked compression

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も現実的な範囲のデータに対して適用することが可能になり，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

[1] 北海道大学大学院 情報科学研究科
Kita 14-jo, Nishi 9-chome, Kita-ku 060-0814, Sapporo, Japan
[2] 日本学術振興会特別研究員 DC
[a)] k_sekine@ist.hokudai.ac.jp
[b)] sasakawa@ist.hokudai.ac.jp
[c)] syoshid@ist.hokudai.ac.jp
[d)] kida@ist.hokudai.ac.jp

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 [14] 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 [7] 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．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．我々の手法では，まず，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する．そして，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．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 [3], [6]．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．約 2.2GB の英文テキストに対する実験の結果，ブロックサイズが 128MB のとき，符号語長（辞書サイズ）が 20 付近で，かつ共有辞書の割合が辞書サイズの半分のときに圧縮率が最も良くなることが判明した．また，提案手法は，固定長の符号を用いているのにも関わらず，パラメータの組み合わせによっては Bzip2 に匹敵する圧縮率（約 30%）に達することが示された．

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 [16] や，LZW 法 [15]，Bisection [4] 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 [5], [7], [9], [10]．それらのうち，特に Re-Pair [7] や，SEQUITUR [9] は圧縮率に優れている．また，Maruyama ら [8] 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 [6] や，Kida [3] がそれぞれ独立に，接尾辞木 [13] 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** [11] 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 [12] 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，論文 [12] では，Tunstall 符号の 100 倍程度の時間を要することが示されてる．この手法は，Gzip 並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分かっている．実際に，この手法は Tunstall 符号の 100 倍程度の時間がかかることを彼らは示している [12].

Wan と Moffat [14] 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 [7] と，提案手法の Blocked-Re-pair-VF について説明する．

3.1 Re-Pair アルゴリズム

Re-Pair アルゴリズムは文脈自由文法に基づく，オフラインの文法圧縮アルゴリズムである．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．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：

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

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

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，生成規則の集合 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 提案手法

本論文の提案手法である，Blocked-Re-pair-VF アルゴリズムは，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，L，s，ℓ をそれぞれ，ブロック長，共有辞書サイズ，符号語長とする．このアルゴリズムは L，s，ℓ を入力として受け取り，一方，入力テキスト T は非負の整数 $\{0, \ldots, |\Sigma| - 1\}$ 上の列で表されているものとする．Blocked-Re-pair-VF は，ブロック間で共有する辞書（**共有辞書**）の構築と，ブロック毎に固有な辞書（**ローカル辞書**）の構築の 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．

入力テキストのうち，最初のブロック $t := T[0..L-1]$ から共有辞書を作成する．以下を共有辞書のサイズが s となるまで繰り返す：

(1) 現在のブロック t 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) 現在のブロック t のすべての 2-gram (α, β) を新しい記号に置き換える．

もし，現在のブロック内のすべての 2-gram がユニークになったら，次のブロックを t として，このブロックについて，現時点での共有辞書のエントリで記号の置換を行ってから，共有辞書の作成を続ける．

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．

4. 実験

Blocked-Re-pair-VF の有用性を検証するための計算機実験をおこなった．

4.1 実験環境と手法

実験は，以下の計算機上で行った．

- CPU : Intel Core i7 processor 2.8GHz
- メモリ : 8GB

- OS : Ubuntu OS 12.04

実験で用いたデータは，*Pizza & Chili corpus*[*1] から取得した英文テキストデータを用いた．このテキストデータの大きさは約 2.2GB であり，アルファベットサイズは 239 である．

まず，提案手法との比較のため，同じデータに対して，既存の手法を適用し，圧縮時間と圧縮率を計測した．比較に用いたのは，Gzip[*2]と, Bzip2[*3], PPMD の 3 つである．PPMD は Prediction by Partial Matching [2] の実装の一つである．プログラムはそれぞれ，*Pizza & Chili corpus* にある gzip, bzip2, および ppmdi を使用した．gzip と bzip2 は，-1 オプションと-9 オプション適用時，ppmdi は-l 0 オプションと-l 9 オプション適用時でそれぞれ比較した．-1 と-l 0 オプションは，圧縮時間を優先し，-9 と-l 9 オプションは圧縮率を優先するオプションである．

表 1 比較手法を *english* に適用した際の圧縮時間と圧縮率.

手法	圧縮率 (%)	圧縮時間 (秒)
gzip -1	42.98	41.201
gzip -9	36.01	193.189
bzip2 -1	31.35	192.539
bzip2 -9	27.05	200.939
ppmdi -1	28.64	314.222
ppmdi -9	23.93	392.932

結果を表 1 にまとめた．Gzip の-1 オプション適用時は，他の手法に対して，圧倒的に短い圧縮時間を達成している．Gzip の-9 オプション，Bzip2 の-1 と-9 オプション適用時は，ほぼ同等の圧縮時間で推移しているが，Bzip2 の-9 オプション適用時がその中でも比較的優秀な圧縮率を実現している．また，PPMD の-l 9 オプション適用時は，最も長い圧縮時間となっているが，同時に最も良い圧縮率を達成している．

4.2 Blocked-Re-pair-VF における実験結果

次に，提案手法に対し，様々なパラメータを与え，その挙動を調査した．提案手法の Blocked-Re-pair-VF は C 言語で実装し，コンパイラは GCC version 4.4 を用いた．実験では，提案手法に対し，ブロックサイズ L，符号語長 ℓ，全体辞書サイズに対して共有辞書が占める割合 s の三つのパラメータを変化させながら，圧縮時間と，圧縮率をそれぞれ計測した．圧縮時間と圧縮率の結果をそれぞれ，表 2，表 3 にまとめた．

4.2.1 ブロックサイズを変化させたときの挙動

表 2 を見ると，ブロックサイズを大きくするほど、圧縮時間は長くなっていることがわかる．これに関して，アルゴリズムの各段階での処理時間をはかったところ，ブロックサイズを大きくするほど，特に，最初のブロックでの共有辞書の作成に時間がかかっていることがわかった．更に，ブロックサイズを大きくするにつれ，ブロック内のラ

[*1] *Pizza & Chili corpus* : `http://pizzachili.dcc.uchile.cl/texts.html`
[*2] Gzip : `http://www.gzip.org/`
[*3] Bzip2 : `http://www.bzip.org/`

表 2 圧縮時間のパラメータによる変化．各表の縦軸は符号語長 ℓ，横軸はブロックサイズ L（単位は MB）である．また，s は全体辞書サイズに対して共有辞書が占める割合である．圧縮時間の単位は秒として与えている．

$s = 0/8$

	32	64	128
15	**535.8**	542.3	546.1
16	560.8	569.3	576.0
17	582.0	593.9	602.8
18	598.7	610.7	623.0
19	614.9	627.2	635.8
20	617.4	640.3	653.9
21	619.5	645.0	666.8
22	620.9	645.7	671.8

$s = 1/8$

	32	64	128
15	**534.8**	540.3	547.3
16	562.9	567.5	576.1
17	581.2	593.2	603.2
18	600.7	612.2	622.5
19	612.5	626.5	640.3
20	620.5	640.5	655.4
21	620.5	653.1	669.3
22	621.1	646.9	672.7

$s = 2/8$

	32	64	128
15	**535.5**	538.0	546.6
16	559.4	568.4	575.0
17	582.0	593.3	602.0
18	597.4	610.5	622.8
19	614.6	627.2	640.3
20	618.8	640.4	653.6
21	623.8	648.2	667.9
22	624.8	649.2	675.6

$s = 3/8$

	32	64	128
15	**533.0**	538.4	545.8
16	558.3	569.2	576.8
17	581.5	591.9	601.1
18	611.0	611.4	622.8
19	611.0	624.9	639.6
20	618.6	636.4	654.7
21	625.0	645.4	667.1
22	625.2	649.1	674.7

$s = 4/8$

	32	64	128
15	**532.0**	536.8	546.3
16	558.4	566.1	578.7
17	580.5	589.2	600.3
18	595.1	609.0	620.6
19	609.8	625.0	638.1
20	618.8	635.3	654.1
21	625.9	645.6	664.5
22	626.6	650.2	672.3

$s = 5/8$

	32	64	128
15	**530.1**	536.9	544.5
16	556.5	564.8	573.7
17	576.9	587.9	599.2
18	593.8	606.2	621.6
19	605.7	624.8	638.3
20	617.3	632.3	650.2
21	622.1	645.1	665.9
22	628.5	652.4	670.7

$s = 6/8$

	32	64	128
15	**527.9**	533.8	544.1
16	553.0	563.1	572.7
17	573.7	587.1	598.5
18	590.2	604.2	620.1
19	603.7	617.8	634.1
20	609.7	628.2	657.3
21	622.1	635.7	658.1
22	629.0	648.5	666.5

$s = 7/8$

	32	64	128
15	**524.6**	531.3	542.1
16	548.4	559.9	570.4
17	570.0	584.1	596.9
18	582.1	601.3	616.3
19	595.9	625.7	632.2
20	602.9	622.6	644.7
21	616.2	630.4	650.1
22	627.0	645.1	661.9

表 3 圧縮率のパラメータによる変化．各表の縦軸は符号語長 ℓ，横軸はブロックサイズ L（単位は MB）である．また，s は全体辞書サイズに対して共有辞書が占める割合である．圧縮率の単位は%として与えている．

$s = 0/8$

	32	64	128
15	33.93	34.14	34.32
16	32.65	32.73	32.85
17	32.01	31.77	31.75
18	32.26	31.40	31.06
19	33.79	31.88	**30.93**
20	35.51	33.41	31.54
21	37.29	35.06	33.02
22	39.06	36.73	34.59

$s = 1/8$

	32	64	128
15	34.14	34.60	34.87
16	32.81	33.16	33.36
17	32.04	32.12	32.20
18	32.06	31.61	31.41
19	33.19	31.82	**31.11**
20	34.32	32.95	31.47
21	35.41	33.99	32.52
22	36.85	35.04	33.47

$s = 2/8$

	32	64	128
15	34.20	34.65	34.89
16	32.81	33.18	33.38
17	31.95	32.10	32.19
18	31.81	31.48	31.35
19	32.66	31.48	**30.94**
20	33.74	32.41	31.11
21	35.17	33.45	31.94
22	36.85	34.90	32.94

$s = 3/8$

	32	64	128
15	34.28	34.72	34.93
16	32.86	33.23	33.41
17	31.92	32.10	32.21
18	31.63	31.40	31.32
19	32.36	31.26	**30.82**
20	33.53	32.11	30.84
21	35.17	33.31	31.68
22	36.40	34.90	32.94

$s = 4/8$

	32	64	128
15	34.43	34.82	34.99
16	32.96	33.31	33.47
17	31.97	32.14	32.24
18	31.55	31.37	31.32
19	32.16	31.19	30.76
20	33.55	31.91	**30.66**
21	35.17	33.31	31.46
22	35.93	34.90	32.93

$s = 5/8$

	32	64	128
15	34.63	34.95	35.07
16	33.14	33.43	33.56
17	32.08	32.25	32.31
18	31.59	31.43	31.36
19	31.99	31.20	30.77
20	33.56	31.83	**30.71**
21	35.19	33.34	31.49
22	35.47	34.33	32.92

$s = 6/8$

	32	64	128
15	34.92	35.15	35.19
16	33.44	33.62	33.68
17	32.35	32.43	32.44
18	31.91	31.62	31.48
19	32.22	31.43	30.89
20	33.62	31.94	**30.88**
21	34.80	33.37	31.61
22	34.92	34.06	32.93

$s = 7/8$

	32	64	128
15	35.40	35.46	35.39
16	33.95	33.95	33.89
17	32.90	32.79	32.67
18	32.58	32.03	31.73
19	32.90	31.93	**31.18**
20	33.91	32.52	31.21
21	34.40	33.54	32.07
22	34.91	33.94	32.85

ンダムアクセスによるキャッシュミスが原因と思われる速度低下が起こっている．

また，表 3 をみると，符号語長が短い時には，ブロックサイズを大きくするほど圧縮率が悪化して，反対に，符号語長が長いときには，ブロックサイズを大きくするほど圧縮率が良くなっていることがわかる．

まず，圧縮率が最も良くなる時に比べ，符号語長が短い時について考察する．ブロックサイズが小さいと，各ブロックで出現する 2-gram の種類が少なくなる．この時は，辞書に登録するエントリ数が少なくてすむので，短い符号語長でもテキストを充分に圧縮することが出来る．一方，ブロックサイズが大きい時は，各ブロックで出現する 2-gram の種類が多くなる．この時に，符号語が短いと，辞書エントリ数が足りなくなり，テキストがうまく圧縮できなくなる．同様に，符号語長が長いときを考察する．ブロックサイズが小さい時には，辞書に登録される 2 グラムの種類に比べて符号語長が長いため，使用されない符号語が数多く存在することになる．したがって，各符号語が必要以上に長くなってしまい，圧縮テキストのサイズも大きくなるため，圧縮率が悪化する．一方，ブロックサイズを大きくすると，各ブロックに出現する 2-gram の種類数は増えるため，圧縮率は改善される．まとめると，ブロックサイズと符号語長はうまくバランスさせる必要があると言え，今回のテストデータに対しては 128MB のブロックサイズに対して，符号語長 20 ビット付近が最も良くなっている．

4.2.2 共有辞書サイズを変化させたときの挙動

表 2 から，共有辞書サイズを大きくするほど，圧縮時間は短くなることがわかる．この理由について考察する．共有辞書サイズを大きくすると，各ブロックのローカル辞書は小さくなる．したがって，各ブロックにおいてローカル辞書を作成するために要する時間が減るため全体の圧縮時間が削減されていると考えられる．また，表 3 より，圧縮率は，共有辞書サイズに対して一様な動きをせず，おおよそ共有辞書の割合が 50%のところで最も小さくなることが観察できる．共有辞書サイズが小さい時には，各ブロックでは大きなローカル辞書が作られるため，合計の辞書サイズは大きくなるが，各ブロックのテキストの性質に合わせた圧縮が可能になる．一方で，共有辞書サイズが大きい時には，辞書の大部分が共有辞書になるため，辞書サイズは小さくなるが，ローカル辞書サイズが各ブロックの性質を捉えるには不十分な大きさになってしまい，テキストを充分に圧縮することができなくなる．よって，上記の 2 つの要素のバランスのとれた共有辞書サイズのとき，圧縮率が最も良くなっていると考えることができる．

4.2.3 符号語長を変化させたときの挙動

表 2 から，符号語長が長くなる程，圧縮時間は長くなっていることがわかる．これは，符号語長が長くなると，生成される辞書のサイズも大きくなるため，辞書の生成に時間がかかるためである．また，表 2 から，圧縮率は，符号語長に対して一様な動きをせず，適切な長さで最小となることがわかる．符号語長が短いと，辞書サイズが小さくなるが，辞書のエントリ数が小さくなり，テキストの圧縮が

うまくいかなくなる．一方，符号語長が長いと圧縮されやすくなるが，辞書サイズが大きくなってしまう．このため，両者のバランスがとれた所で最小となると考えることができる．今回，ブロック長 128MB，符号語長 20 ビット，共有辞書の割合が 50%のときが最良であり，30.66%の圧縮率を達成している．

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 を提案した．提案手法では，良い圧縮率を実現するため，辞書の一部を他のブロックと共有する手法を採用した．実験では，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．今回の手法では，共有辞書が最初のブロックのみに深く依存して構築される．これらの問題を解決する新しい手法に関して調査を行うことが今後の課題である．

また，今回は，提案手法と Re-Merge アルゴリズム [14] 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．

謝辞

本研究は JSPS 科研費 23700002 および 24240021 の助成を受けたものです．

参考文献

[1] Burrows, M. and Wheeler, D. J.: A block-sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation, Palo Alto, California (1994).
[2] Cleary, J. and Witten, I.: Data Compression Using Adaptive Coding and Partial String Matching, *Communications, IEEE Transactions on*, Vol. 32, No. 4, pp. 396 – 402 (online), DOI: 10.1109/TCOM.1984.1096090 (1984).
[3] Kida, T.: Suffix Tree Based VF-Coding for Compressed Pattern Matching, *Proc. of Data Compression Conference 2009 (DCC 2009)*, p. 449 (2009).
[4] Kieffer, J. C., E.-H. Yang, G. N. and Cosman, P.: Universal Lossless Compression via Multilevel Pattern Matching, *IEEE Trans. Inform. Theory*, Vol. 46, No. 4, pp. 1227–1245 (2000).
[5] Kieffer, J. C. and Yang, E.-H.: Grammar-Based Codes: a New Class of Universal Lossless Source Codes, *IEEE Trans. on Inform. Theory*, Vol. 46, No. 3, pp. 737–754 (2000).
[6] Klein, S. T. and Shapira, D.: Improved Variable-to-Fixed Length Codes, *Proc. of the 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008)*, pp. 39–50 (2008).
[7] Larsson, N. J. and Moffat, A.: Off-line dictionary-based compression, *Proceedings of the IEEE*, Vol. 88, No. 11, pp. 1722–1732 (2000).
[8] Maruyama, S., Tanaka, Y., Sakamoto, H. and Takeda, M.: Context-Sensitive Grammar Transform: Compression and Pattern Matching, *Proc. of 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008)*, pp. 27–38 (2008).
[9] Nevill-Manning, C., Witten, I. and Maulsby, D.: Compression By Induction of Hierarchical Grammars, *Proc. of the Data Compression Conference 1994 (DCC '94)*, IEEE, pp. 244–253 (1994).
[10] Sakamoto, H., Kida, T. and Shimozono, S.: A Space-Saving Linear-Time Algorithm for Grammar-Based Compression, *String Processing and Information Retrieval*, Lecture Notes in Computer Science, Vol. 3246, Springer Berlin / Heidelberg, pp. 218–229 (2004).
[11] Tunstall, B. P.: Synthesis of noiseless compression codes, PhD Thesis, Georgia Institute of Technology, Atlanta, GA (1967).
[12] Uemura, T., Yoshida, S., Kida, T., Asai, T. and Okamoto, S.: Training parse trees for efficient VF coding, *Proc. of the 17th international conference on String processing and information retrieval (SPIRE 2010)*, pp. 179–184 (2010).
[13] Ukkonen, E.: On-line construction of suffix trees, *Algorithmica*, Vol. 14, No. 3, pp. 249–260 (1995).
[14] Wan, R. and Moffat, A.: Block merging for off-line compression, *J. Am. Soc. Inf. Sci. Technol.*, Vol. 58, No. 1, pp. 3–14 (online), DOI: 10.1002/asi.v58:1 (2007).
[15] Welch, T. A.: A Technique for High Performance Data Compression, *IEEE Comput.*, Vol. 17, pp. 8–19 (1984).
[16] Ziv, J. and Lempel, A.: Compression of Individual Sequences via Variable-length Coding, *IEEE Trans. on Inform. Theory*, Vol. 24, No. 5, pp. 530–536 (1978).