Vol.2014-AL-148 No.8
2014/6/13

簡潔索引を用いたVF符号上の部分文字列抽出

笹川 裕人[1]　関根 渓[1]　吉田 諭史[1]　喜田 拓也[1]

概要：本稿では，可変長-固定長符号 (VF 符号) により符号化された圧縮テキストに対する，高速な部分文字列抽出法を提案する．提案手法では，圧縮テキストに対して，符号語の境界に対応する元テキストの位置を格納した簡潔索引を付加することで，圧縮テキストから部分文字列を抽出する問題を平均 $O(N/n+\ell)$ 時間で解く．ここで，N と，n,ℓ は，それぞれ元テキスト長，圧縮テキスト長，抽出する部分文字列長である．計算機実験では，提案手法を Re-Pair-VF アルゴリズム上で実装し，高速に部分文字列抽出問題を解けることを実証した．

Simple Substring Extraction for Grammar Compressed Text using VF coding and Succinct Index

Abstract: In this paper, we propose a simple method solving the substring extraction problem for variable-to-fixed length codes (VF codes) by adding an auxiliary succinct index structure. The method solves the problem in $O(\frac{N}{n}+\ell)$ time in the average case, where N, n, and ℓ, are the original text length, the encoded text length, and the length of the target substring, respectively. We implemented our methods with exploiting Re-Pair-VF and showed that our methods run fast in actual.

1. 序論

VF 符号 (*Variable-to-fixed length* 符号) は，入力テキストを連続する可変長の部分文字列（ブロックと呼ぶ）に分割し，それぞれのブロックに固定長の符号語を割り当てる符号化方式である．この方式では，全ての符号語が同じ長さの符号長である為，符号化されたデータから任意の符号語の抽出が容易に行なえる．このような性質をもつことから，VF 符号は，圧縮データを復号することなく直接扱うことに適した符号化方式であると言える．

最近の研究において，圧縮パターン照合や，圧縮データ上のデータマイニングなどの観点から，VF 符号は再評価されている [2,4,6,10,12]．これに関して，吉田と喜田ら [11] は，Re-Pair アルゴリズム [5] と呼ばれる文法圧縮を基にした VF 符号である Re-Pair-VF を提案している．文法圧縮は，与えられたテキストを唯一に導出する文脈自由文法を構築し，その文法自体を符号化することでデータ圧縮を達成する手法である [3]．なかでも Re-Pair アルゴリズムは，きわめて優れた圧縮率を達成できる文法圧縮である．実際，Re-Pair-VF は，英文テキストデータに対して，gzip よりも良い圧縮率を達成する．

本論文では，VF 符号化されたテキストから部分文字列を抽出する問題を考える．この問題では，入力として符号化されたテキスト $\mathcal{E}(T)$ と，位置 pos，長さ ℓ が与えられたとき，文字列 $T[pos],\ldots,T[pos+\ell-1]$ を出力する．以降では，この問題を**部分文字列抽出問題**と呼ぶことにする．符号化テキストを復号してから，テキストを先頭から走査する素朴な手法では，所望の部分文字列を抽出するまでに $\Theta(pos+\ell)$ 時間かかる．

関連研究として，Bille ら [1] は，文法圧縮の手法に基づいた圧縮テキスト上で，部分文字列抽出問題を $O(\log N+\ell)$ 時間で解くアルゴリズムを与えている．ここで，N は元のテキスト T の長さである．丸山ら [7] は，オンラインで文法圧縮を行う FOLCA という手法を提案し，その中で，部分文字列抽出問題を $O((\log N+\ell)\log g/\log\log g)$ 時間で解く手法を与えている．ここで，g は文法のサイズである．これらの手法は優れた最悪時計算量を達成するが，圧縮テキストデータ全てを主記憶上にロードする必要があり，圧縮テキストデータ全体が主記憶に収まらない大きさのとき

[1] 北海道大学大学院情報科学研究科
Hokkaido University, Graduate School of Information Science and Technology, {sasakawa, tmasaki, kida}@ist.hokudai.ac.jp

問題となる．

我々は，この問題に対して，VF 符号と索引構造を組み合わせたよりシンプルな手法を提案する．索引構造は，テキスト T 上のブロック境界の位置を格納した完備辞書を用いる．この索引構造により，$T[pos]$ が含まれる符号語に定数時間でアクセスすることが出来る．該当する符号語を特定したあとは，その符号語を部分的に復号することで，指定された部分文字列を取り出す．提案手法は，部分文字列抽出問題を $O(N+\ell)$ 最悪時計算時間，もしくは，$O(\frac{N}{n}+\ell)$ 平均時計算時間で解く．ここで，n は，VF 符号における圧縮テキストサイズである．ここで，$\frac{N}{n}$ は，圧縮率の逆数に当たるものであり，実際にはほぼ定数と見ることが出来る．

提案手法は，索引構造と圧縮データの辞書部分のみをメモリ上にロードするだけで動作し，圧縮テキスト全体をロードする必要は無い．ただし，索引構造を構築する手間と保存するコストが余分にかかるため，それらは十分に小さいことが望ましい．今回の計算機実験により，提案手法が，FOLCA に対してほぼ同等の圧縮率を達成しつつ，数倍以上高速に部分文字列抽出問題を解くことを確認した．

2. 準備

2.1 Re-Pair アルゴリズム

形式的には，Re-Pair アルゴリズム（以降では単に Re-Pair と書く）は 4 つ組 (Σ, V, σ, R) で表される文脈自由文法 (CFG) G を一つ生成する．ここで，$\Sigma = \{a_0, a_1, \cdots, a_{|\Sigma|-1}\}$ は終端記号の集合であり，$V = \{\alpha_0, \alpha_1, \cdots, \alpha_{|V|-1}\}$ は非終端記号の集合である．また，$\sigma \in V$ は開始記号，R は $V \times (\Sigma \cup V)^*$ 上の有限部分集合である．R の要素は生成規則と呼ばれ，$\alpha \in V, \gamma \in (\Sigma \cup V)^*$ に対し $\alpha \to \gamma$ のように記述される．

Re-Pair によって生成された CFG G は次のような生成規則を含む．

$$\sigma \to \alpha_{i_0}\alpha_{i_1}\cdots\alpha_{i_{m-1}} \quad (\forall i_k \in \{0, \cdots, |\Sigma|+|V|-2\}),$$

$$\alpha_i \to \begin{cases} a_i & (0 \le i < |\Sigma| \text{ の場合}), \\ \alpha_j\alpha_k \quad (0 \le j, k < i) & (i \ge |\Sigma| \text{ の場合}). \end{cases}$$

ここで，生成規則の右辺に出現するバイグラムはすべて異なる．Re-Pair は入力テキスト T 中に出現するバイグラムの中から最頻出のもの $\alpha\beta$ を一つ選び，その最頻のバイグラムの出現を左から順番に新しい記号 A に置き換える．そして，生成規則 $A \to \alpha\beta$ を R に追加する．この手続きを，テキスト上のすべてのバイグラムの頻度が 1 になるまで繰り返す．上記の置換え手続き後のテキスト T' を開始記号 σ から生成する生成規則 $\sigma \to T'$ を R に加えることで，元のテキスト T を唯一に導出する CFG G を得る．最終的には，G に適当な符号化を行い圧縮データを得る．

Larsson と Moffat ら [5] によって，Re-Pair は入力テキスト長 n に対し，$O(n)$ 時間で CFG の構築を完了することが示されている．その計算時間を達成するためには，入力テキストを，同一のバイグラムどうしを前後でリンクさせた双方向連結リストに変換する必要がある．さらに，任意のバイグラムを保持する記述子へ $O(1)$ 時間でアクセスするためのハッシュテーブルとバイグラムの頻度を管理するための優先度付きキューを必要とする．

今回の実験で用いた Re-Pair-VF [11] は，上述した Re-Pair を利用しつつ，文法 G に固定長の符号化を用いる VF 符号化の手法である．Re-Pair-VF では，すべての生成規則が $\log(|\Sigma|+|V|)$ ビットの符号語で表現される．ただし，Re-Pair がバイグラムがユニークになるまで生成規則の構築を進めるのに対し，Re-Pair-VF では，固定長符号化した際に，最も効率が良くなる V の時点で構築を打ち切る仕組みを導入している．

2.2 Rank/Select 辞書

簡潔データ構造による Rank/Select 辞書（完備辞書）について説明する．ビット列 B に対する Rank/Select 辞書は次のように定義される．$k \in \{0, 1\}$ に対して，

$$\mathrm{Rank}(B, i, k) = |\{n \in [0 \ldots i] \colon B[i] = k\}|,$$
$$\mathrm{Select}(B, i, k) = \min\{n \in B \colon \mathrm{Rank}(B, n, k) = i\}.$$

すなわち，$\mathrm{Rank}(B, i, k)$ は $B[0 \ldots i]$ 中の k の個数を，$\mathrm{Select}(B, i, k)$ は B 中の k の i 番目の位置を示す．$\mathrm{Rank}(B, i, k)$ と $\mathrm{Select}(B, i, k)$ はそれぞれ，より簡潔に $\mathrm{Rank}_k(B, i)$，$\mathrm{Select}_k(B, i)$ と書くことがある．対象とするビット列 B が文脈から明らかな場合は，さらに記述を簡略化し，$\mathrm{Rank}_k(i)$，$\mathrm{Select}_k(i)$ と書くことにする．

これまで，Rank/Select 操作の処理速度を犠牲にすることなく，ビット列の記憶容量を理論的下限に近づける研究がなされてきた．とりわけ，DARRAY [8] と SDARRAY [8]，RRR [9] の三つはよく知られた完備辞書の実現手法である．DARRAY は，1 の個数がきわめて多い（密な）ビット列に適している．一方，1 が疎なビット列に対しては，SDARRAY が適している．RRR は 1 と 0 の出現が偏っている場合に適していることが知られている．極端な例でいえば，長さ $2n$ ビットの先頭半分が 0 で残りが 1 で埋まっているようなビット列の場合である．DARRAY と RRR は，Rank/Selcet 操作のどちらもが定数時間で行える．一方，SDARRAY は，Select 操作は定数時間であるが，Rank 操作は最悪時 $O(\log(n/m))$ の時間を必要とする．ここで，m は，ビット列 B 中に出現する 1 の個数である．ただし，平均的には定数時間で，かつ非常に高速である．

3. 提案手法

本節では，Re-Pair-VF 圧縮テキスト上で高速な直接アクセスを行う手法を提案する．まず，用いる索引構造を，

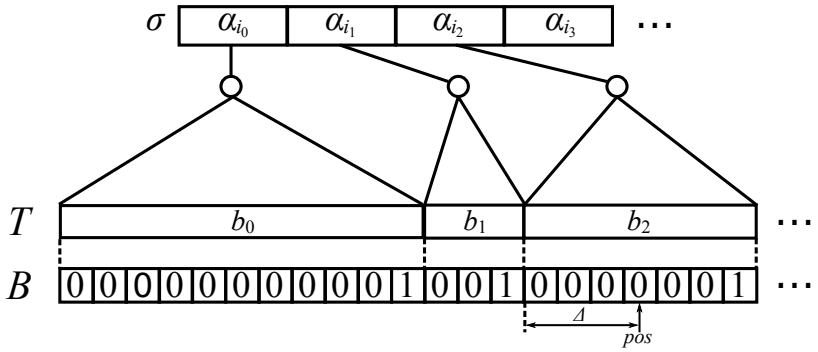

図 1　提案手法で用いる索引構造の例．ここで，$\sigma = \alpha_{i_0}\alpha_{i_1}\alpha_{i_3}\ldots$ は VF 符号の列である．b_j は T 中の j 番目のブロックで，α_{i_j} を展開した文字列に対応する．Δ は，$j = \mathrm{Rank}_1(\mathrm{pos})$ に対応するブロック b_j の先頭位置から pos までのオフセットである．

以下の手順で構築する．

(1) Re-Pair-VF で入力テキストを圧縮し，Re-Pair のバイグラムの置き換え部分の生成規則の集合（以降では辞書と呼ぶ）D と圧縮後の系列 σ を構成する．

(2) 次に，以下に示す長さ n のビット列 B を構築する．

$$B[i] = \begin{cases} 1 & (T[i] \text{ がブロックの最後の文字の場合}), \\ 0 & (\text{それ以外}). \end{cases} \tag{1}$$

図 1 に σ, T, B の対応関係を表す例を示す．

(3) ビット列 B から $o(n)$ ビットを用いて完備辞書 I を構築する．

最終的な圧縮データは，D と σ に I を加えたものとなる．実際にデータアクセスを行う際は，D と I のみメモリ上にロードしておき，σ は二次記憶装置に置いたまま，必要な部分のみを読み込むことで使用するメモリ量を削減することができる．

圧縮データへの直接アクセスは次の手順で行う．

(1) 元テキスト上の位置 pos が与えられたとき，$T(pos)$ を含む符号語の位置は $\mathrm{Rank}_1(\mathrm{pos})$ で求められる．なぜならば，$j = \mathrm{Rank}_1(i)$ は，ビット列 B の i 番目までの 1 の個数を返すので，I に対して rank 操作を行えば pos までの符号語の個数を得ることができる．

(2) 位置 $j = \mathrm{Rank}_1(\mathrm{pos})$ について，σ から α_{i_j} を取り出す．

(3) 部分文字列開始位置までのオフセット Δ は次のようにして求められる．

$$\Delta = pos - \mathrm{Select}_1(j) \tag{2}$$

(4) 辞書 D を用いて取り出した符号語を（部分的に）展開し，$T[pos]$ を得る．

VF 符号でデータを圧縮していれば，各符号語の長さは一定である．そのため，目的の符号語へのアクセスは，配列へのアクセスと同じように定数時間で行える．したがって，目的のデータ位置へのアクセスは，符号語の先頭位置から pos までのオフセットを Δ とすると，$O(\Delta)$ 時間で行える．最悪時は，$|\sigma| = O(1)$ まで圧縮される場合であり，このとき $O(pos)$ 時間となる．ただし，平均時には，$\Delta = O(\frac{N}{n})$ であり，これはほぼ定数である．

4. 実験

提案手法の有用性を評価するため，いくつかのデータセット上で計算機実験を行った．実験は，以下の計算機上で行った．

- CPU: 3.6GHz Intel Xeon processor
- メモリ: 32GB
- OS: Debian GNU/Linux 6.0.8

データは，*Pizza & Chili Corpus*[*1] から取得した *dna* と，*proteins*，*english* を用いた．なお，*proteins* と，*english* については，先頭の 500 MB のみを利用した（それぞれ *proteins.500MB* と，*english.500MB* と表記する）．提案手法は，文法圧縮手法として Re-Pair-VF を用い，前節で説明したビット列 B を，RRR [9]，SDARRAY [8]，DARRAY [8] をそれぞれ用いて保持する手法 RVF+RRR，RVF+SDA，RVF+DA を用意した．また，比較手法として，FOLCA [7] と，Re-Pair-VF に対して，索引構造を付けずに前から走査して部分文字列を抽出する素朴な手法 RVF w/o index を用いた．

まず，上記の各手法に対して，圧縮率，圧縮時間，部分文字列抽出にかかる時間を比較した．圧縮時間は，Re-Pair-VF によるデータの圧縮時間と，索引構造の構築時間の合計である．圧縮時間の実験結果を表 1 に示す．全てのデータセットについて，FOLCA が最も速く，提案手法の中では，RVF-RRR が最も高速であった．圧縮率は，提案手法に対しては，(圧縮データサイズ + 索引サイズ) / (元データサイズ) により計算され，FOLCA については，(圧縮データサイズ) / (元データサイズ) により計算される．圧縮率の実験結果を表 2 に示す．提案手法の圧縮率は，RVF w/o index よりも索引構造の分悪くなっているのが分かる．dna と protains に対しては，FOLCA が，english に対しては

[*1] http://pizzachili.dcc.uchile.cl/texts.html

表 1　圧縮時間の比較 (秒)

Method	dna	english.500MB	proteins.500MB
RVF w/o index	98.794	151.740	169.339
FOLCA	72.798	116.101	114.823
RVF+DA	106.262	161.824	180.002
RVF+SDA	109.864	163.493	181.687
RVF+RRR	103.541	159.260	177.435

表 2　圧縮率の比較 (%)

Method	dna	english.500MB	proteins.500MB
RVF w/o index	28.43	30.53	46.27
FOLCA	40.69	47.25	50.34
RVF+DA	47.83	47.62	63.78
RVF+SDA	60.86	45.54	62.39
RVF+RRR	41.05	38.60	54.17

RVF+RRR が最も良い圧縮率となった．提案手法の中では，全てのデータセットに対して RVF+RRR が最も良い圧縮率を達成した．また，RVF+SDA は，dna に対してもっと悪く，english と protein については，RVF+DA が最も悪い圧縮率となった．これは，ビット列 B の密度が原因であると考えられる．dna におけるビット列の密度は 0.28 であり，protein の 0.13 や，english の 0.11 に比べ高いため，SDARRAY の索引部分のサイズが悪化したと考えられる．

次に部分文字列抽出にかかる時間についての比較を行った．ここでは，先頭から 0 から 400,000,000 番目の各位置に対して，長さ 10 文字の部分文字列に対する抽出時間を計測した．計測した時間は，クエリを与えてから，データをロードする時間も含めている．実験結果を図 2 から図 4 に示す．図から，RVF w/o index は，抽出位置に対して線形に時間が増加している．一方，それ以外の手法は，抽出位置によっての時間の変化はほとんど見られなかった．また，全ての提案手法は，FOLCA と比較すると，今回のデータセットにおいて 10〜50 倍程度高速に動作している．これは，提案手法が圧縮データの一部のみを読み出すのに対して，FOLCA は圧縮データの全てを読み出す必要があるためであると考えられる．提案手法は，最初に，辞書とビット列 B のみを主記憶に読み出し，抽出を行う対象となる符号語を B に対して Rank，Select 演算を行う事により求める．また，VF 符号を用いる事により，対象となる符号語に $O(1)$ 時間でアクセスする事が出来る事も高速化の要因である．提案手法の中では，RVF+DA と，RVF+SDA の結果はほぼ同一であり，RVF+RRR は，それらより 2〜8 倍遅い．この理由は，RRR は読み出す際に必要な追加のデータ構造の構築に時間がかかるためである．

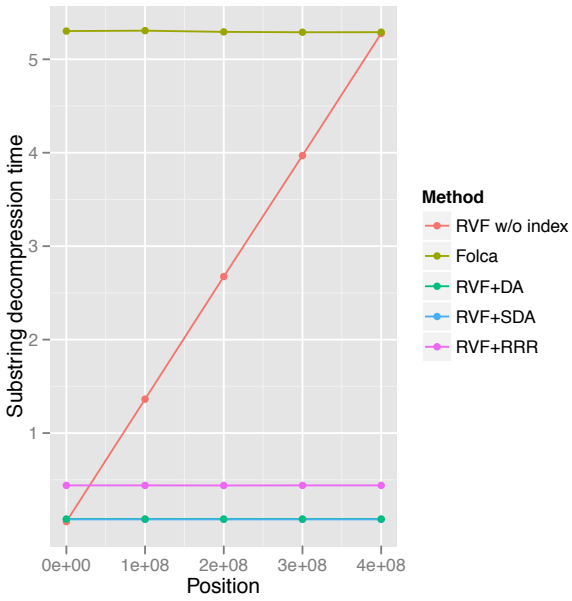

図 2　部分文字列抽出時間の比較 (english)

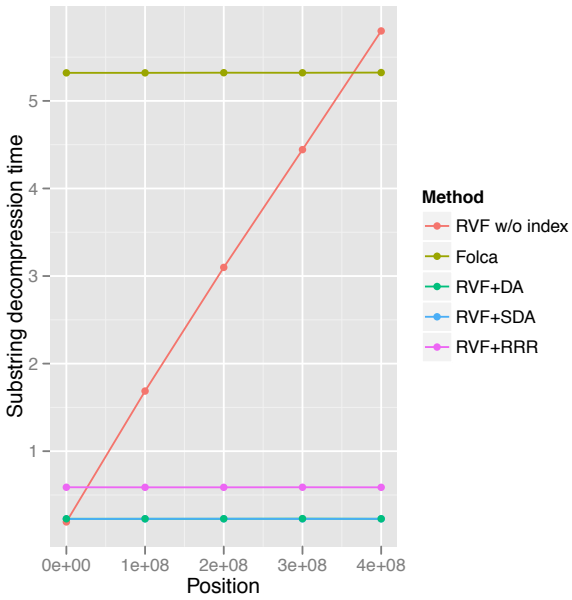

図 3　部分文字列抽出時間の比較 (proteins)

5. 結論

本論文では，圧縮データに対する部分文字列抽出を高速に行う手法を提案した．提案手法は，Re-Pair-VF で圧縮されたデータに対して，符号語の境界情報を格納した索引構造を付加することで，部分文字列の抽出を高速に実行する．計算機実験では，提案手法は，比較手法の FOLCA に対してほぼ同等の圧縮率を達成し，部分文字列抽出については，非常に高速に動作することを確認した．

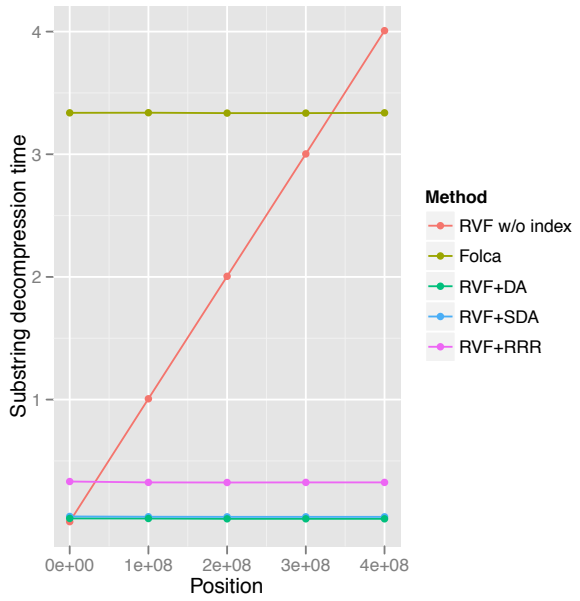

図 4　部分文字列抽出時間の比較 (dna)

参考文献

[1] Bille, P., Landau, G. M., Raman, R., Sadakane, K., Satti, S. R. and Weimann, O.: Random access to grammar-compressed strings, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, SIAM, pp. 373–389 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2133036.2133066⟩ (2011).

[2] Kida, T.: Suffix Tree Based VF-Coding for Compressed Pattern Matching, *Proceedings of Data Compression Conference 2009 (DCC 2009)*, p. 449 (2009).

[3] 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).

[4] Klein, S. T. and Shapira, D.: Improved Variable-to-Fixed Length Codes, *Proceedings of the 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008)*, pp. 39–50 (2008).

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

[6] 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).

[7] Maruyama, S., Tabei, Y., Sakamoto, H. and Sadakane, K.: Fully-online grammar compression, *Proceedings of the 20th international conference on String processing and information retrieval (SPIRE 2013)*, pp. 218–229 (2013).

[8] Okanohara, D. and Sadakane, K.: Practical Entropy-Compressed Rank/Select Dictionary, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX 2007)*, pp. 60–70 (2007).

[9] Raman, R., Raman, V. and Rao, S. S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets, *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 233–242 (online), available from ⟨http://dl.acm.org/citation.cfm?id=545381.545411⟩ (2002).

[10] Yamamoto, H. and Yokoo, H.: Average-Sense Optimality and Competitive Optimality for Almost Instantaneous VF Codes, *IEEE Transactions on Information Theory*, Vol. 47, No. 6, pp. 2174–2184 (2001).

[11] Yoshida, S. and Kida, T.: A Variable-length-to-fixed-length Coding Method Using a Re-Pair Algorithm, *IPSJ Transactions on Databases*, Vol. 6, No. 4, pp. 17–23 (2013).

[12] Yoshida, S., Uemura, T., Kida, T., Asai, T. and Okamoto, S.: Improving Parse Trees for Efficient Variable-to-Fixed Length Codes, *Journal of Information Processing*, Vol. 20, No. 1, pp. 238–249 (2012).