DEIM Forum 2009 C7-5

データ圧縮の理論に基づく効率的な索引構造

田中洋平† 大津祐樹† 岸上直也†† 坂本比呂志†

† 九州工業大学 大学院情報工学研究科
†† 九州工業大学 情報工学部
〒 820-8502 福岡県飯塚市大字川津 680-4
E-mail: {t_youhei, y_ootsu, n_kishiue}@donald.ai.kyutech.ac.jp, hiroshi@ai.kyutech.ac.jp

あらまし 本研究では，アルファベット還元と呼ばれるデータ圧縮法を応用した，圧縮データそのものを索引とする高速・省スペースなデータ構造を提案する．アルファベット還元とは，特別な作業領域を必要としない理論的に最適な圧縮法であり，テキスト中の共通する部分文字列を同じ文字で符号化するという特性がある．本研究では，この特性を索引として応用することで，符号化された文字のみから元の部分文字列の出現を高速に判定する手法を提案する．
キーワード 情報検索, 索引構造, アルゴリズム, データ圧縮

Efficient Indexing Based on Data Compression Theory

Youhei TANAKA†, Yuuki OOTSU†, Naoya KISHIUE††, and Hiroshi SAKAMOTO†

† Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology
†† Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology
Kawazu 680-4, Iizuka-shi, Fukuoka 820-8502
E-mail: {t_youhei, y_ootsu, n_kishiue}@donald.ai.kyutech.ac.jp, hiroshi@ai.kyutech.ac.jp

Abstract We propose an efficient data structure for rapid information retrieval based on the special data compression called *alphabet reduction*. The alphabet reduction is an optimum compression technique without extra memory space and is characteristic in its structure-preserving compression. Using this compression technique we construct an algorithm to detect any pattern in a large text from the compressed index only.
Key words Information retrieval, Index structure, algorithm, data compression

1. はじめに

本研究では，辞書式圧縮法による圧縮アルゴリズムを応用した軽量・高速な索引構造を提案する．本手法では，対象となるテキストの圧縮データそのものを索引として利用している．特に，索引を構築するために入力データ全体を一時的にメモリに格納する必要がないため，作業領域が従来手法と比較して省スペースであり大規模データへの適応が可能である．

辞書式圧縮法では，入力テキストのみを生成する文脈自由文法 (CFG) を考え，そのような CFG のうちできるだけサイズの小さなものを出力することが目的である．この最適化問題は NP-困難であり，入力長 n に対して，最適解に対する $o(\frac{\log n}{\log\log n})$ 近似(注1)は困難であると予想されている．そこでまず，CFG の最小化問題についてのこれまでの研究をまとめる．

これまでに，LZ77 や LZ78 [15], [16] をはじめ，多くの実用的な圧縮法が提案されてきたが，最適解に対するそれらの圧縮の理論的な近似率はいずれも $\Omega(n^{\frac{1}{3}})$ より悪い [9]．最初のよい近似アルゴリズムは，Charikar [2] らによって示された $O(\log\frac{n}{g})$ 近似アルゴリズムである．ここで g は最適解のサイズである．また同時期に，Rytter [11] は，接尾辞木を用いた線形時間 $O(\log\frac{n}{g})$-近似アルゴリズムを構築した．その後，Sakamoto [14] は，BPE (byte pair encoding) を改良した線形時間 $O(\log n)$ 近似アルゴリズムを提案している．

これらによる圧縮は最適解に飛躍的に近づいているが，いずれも接尾辞木 [4] などのテキスト全体に対する $\Theta(n)$ 領域のデータ構造を必要とするため，データサイズの増加が主記憶を圧迫してしまう．例えば [14] の主記憶量は BPE の実装に依存し，そのひとつである Repair [8] は，$5n + 4k^2 + 4k' + \sqrt{n}$ 文字分の主記憶を使用する．ここで k と k' はそれぞれ入力と出力アルファベットのサイズである．これに対して，[12] で提案された圧縮法は，各アルファベットがテキスト中に最初に出現する位置情報のみを使用する単純な手法に基づき，最適解に対する近似率 $O((\log^* n)\log n)$ を保証する．ここで $\log^* n$ は，

(注1)：最小 CFG の $\frac{\log n}{\log\log n}$ 倍未満の解を保証する多項式時間アルゴリズム

$\log\log\cdots\log n > 1$ を満たす log の最大回数を表し，実用的にはほぼ定数(注2)と考えてよい．また前述の理由から，これは準最適な圧縮といえる．

[12] のアルゴリズムは圧縮を行うための作業用領域として，それまでに生成した辞書情報 (すなわち文字の置き換えを表す CFG) 以外の領域を必要としないため，その他の実用的なアルゴリズムと比較しても飛躍的に省スペースであることが保証される．本研究で提案するパターン検索のための索引は，この圧縮アルゴリズムによって生成される CFG に基づいている．

この圧縮アルゴリズムは，テキスト中に現れるパターンを出現位置によらず保存するという性質を持っている．より正確には，パターン P のテキストにおける任意の二つの出現が，xAy および uAv のように圧縮されることを意味する．A はある一文字であり，x, y, u, v は適当な文字列を表す．A のような各文字とそれが符号化している文字列の関係は CFG として記録されている．

ここで，テキスト中に P が出現するには，辞書に A が出現することが必要条件であり，P の残りの接頭辞・接尾辞 (すなわち x, y, u, v) に対して同様の検索を行うことで，辞書へのアクセスのみで P の出現を確定できる．また，A は P の十分長い部分文字列(注3)を符号化していることが保証できるため，上記の再帰的なパターン検索全体をテキスト長にほぼ無関係に行うことができる．本研究ではこの処理を高速に実行するためのアルゴリズムと索引構造を提案する．

2. 圧縮アルゴリズム

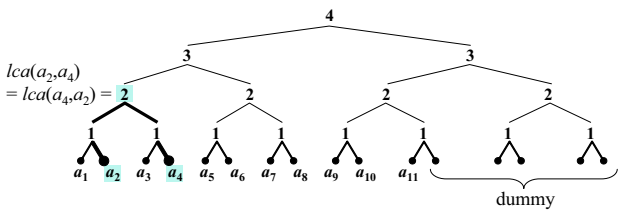
この節では，提案手法の基礎となる圧縮アルゴリズムについて述べる．このアルゴリズムはアルファベット還元と呼ばれる文字の変換規則に基づいている．

2.1 アルファベット還元

文字の集合を Σ で表す．テキスト中にあらかじめ出現する文字とは別に，アルゴリズムが生成する文字は特に変数と呼んで区別する．また，これらの文字全体をまとめてアルファベットと呼ぶ．ある Σ に対して，それらの文字を葉として持つ完全平衡 2 分木を考え，アルファベット木と呼ぶ．図 1 に $|\Sigma| = 11$ の場合のアルファベット木を示す．この木によって，任意の文字の組 (digram) a_ia_j に対して，$lca(i, j)$ を a_i と a_j の最近共通祖先の高さとして定める．

文字 a_i と整数 i を同一視すると，関数 $lca(i, j)$ によって，

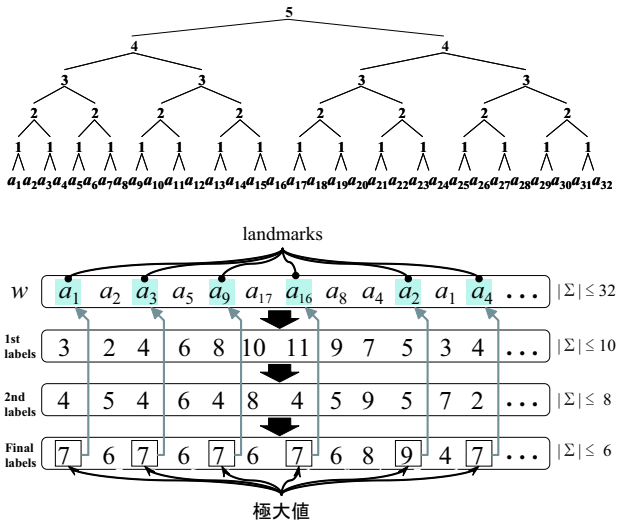
図 1　アルファベット木



(注2)：$n = 2^{65536}$ のような巨大な数に対しても $\log^* n = 5$ に過ぎない

(注3)：P のおよそ半分に相当する長さを持つが詳細な解析は省略する

図 2　アルファベット還元



$\{1, \ldots, n\}$ で表現された文字列が，$\{1, \ldots, \lfloor \log n \rfloor + 1\}$ で構成された同じ長さの整数列に変換される．すなわち，テキスト中の k 番目の文字 a_i は，その左隣の文字 a_j によって決まる整数 $lca(i, j)$ に変換される．ただし，1 番目の文字は左隣の文字を持たないため，あらかじめ決まった値に変換されるとする．この変換をアルファベット還元と呼ぶ [3]．

この還元によって変換された整数列を新たな文字列と見なして同様にアルファベット還元を行うことで，整数列に含まれる整数の種類を十分に小さくできる．この還元の様子を図 2 に示す．整数列に含まれる整数の種類が十分小さく (6 以下に) なるまでこの還元を繰り返したとき，左右の値よりも大きい整数値を取る文字を landmark と呼ぶ．

2.2 圧縮アルゴリズムの概略

図 3 に圧縮アルゴリズムの概略を示し，各 Phase でどのような処理を行うかを説明する．

[Phase1] 与えられたテキスト w を $w = w_1w_2\cdots w_m$ の形に分解する．ここで各 w_i は，同じ文字が連続する a^k の形かそれ以外の不連続形で分解される極大な部分文字列である．例えば $w = ababbbab$ のときの分解は aba, bbb, ab となる．

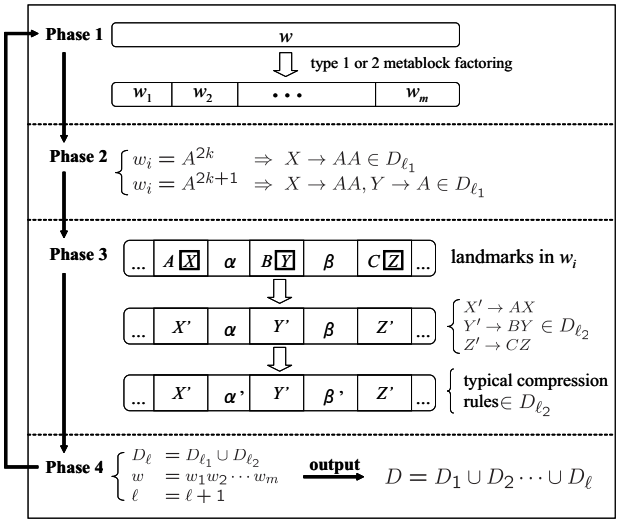
[Phase2] a^k の形の分解に対して，出現するすべての digram を一文字で置き換えて，$A \to aa$ の生成規則を作る．ただし，k が奇数の場合は，最後の一文字はそのまま残される．したがって，a^k は $AA\cdots A$ または $AA\cdots Aa$ のいずれかに変換される．

[Phase3] 不連続形の部分文字列を圧縮する．まず文字列中の landmark を計算し，その隣の文字とで構成される digram を一文字に置き換える．すべての landmark が置き換えられると，次に，それ以外の置き換えられなかった digram を左側優先で一文字に置き換えていく．この場合も，残りの一文字は無視する．

[Phase4] これまでに別々に置き換えた文字列を連結してひとつの文字列にして，Phase1 からの処理を，文字列がそれ以上圧縮できなくなるまで繰り返す．

圧縮アルゴリズム A の近似率 $app(A, w)$ は次の式で定義される．ただし，$|G|$ は CFG のサイズすなわち生成規則の右辺

図 3　圧縮アルゴリズムの概略



の長さの総和を表す.

$$app(A, w) = \max_{w \in \Sigma^*} \left\{ \frac{|G_A(w)|}{|G_{opt}(w)|} \right\}$$

このアルゴリズムの近似率は，与えられたテキスト長を n とすると，$O((\log^* n) \log n)$ であることが示されている [12]．このアルゴリズムによって文字列がどのように圧縮されるかを図 4 に示す．この図で示されている文字列は次のように定義される**フィボナッチ文字列**であり，圧縮アルゴリズムのベンチマークとして利用されている．

$$\begin{cases} f_0 = \text{'a'},\ f_1 = \text{'b'} \\ f_{n+1} = f_{n-1} \cdot f_n \end{cases}$$

例えば，$n = 6$ の場合のフィボナッチ文字列は f_6 = 'abbabbababbab' となる．次の節では，このアルゴリズムによって構築された CFG を利用したパターン検索のための索引を提案する．

3. CFG による索引

前節の圧縮アルゴリズムで構築した CFG G について以下の性質が成り立つ．ここで，$w[i]$ は文字列 w の i 番目の文字を表し，$w[i, j]$ は文字列 w における $w[i]$ から $w[j]$ までの部分文字列を表す $(i \leqq j)$.

[**定理 1**]　$w[i, j] = w[k, \ell] \in \Sigma^*$ とする．このとき，$w[i, j] = w[k, \ell] = x\alpha y$ となる十分に長い部分文字列 α が存在して，α が同じ変数に置き換えられる．

定理 1 で α の長さは十分に長いと説明しているが，これは

図 4　フィボナッチ文字列の圧縮結果

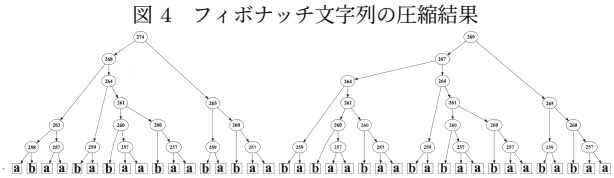
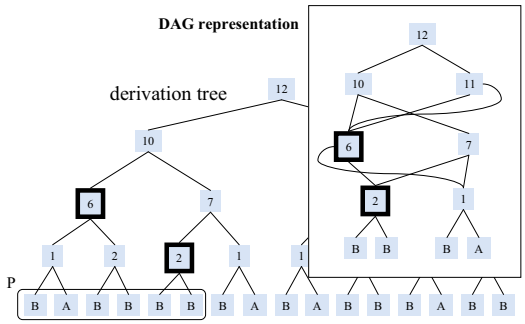


図 5　構文木と DAG



$\log^* n$ を定数と見なすと，$|\alpha| = \Omega(j - i)$ であることを意味している．したがって，α が変数 A に置き換えられているとき，$w[i, j]$ の出現を，A の G における出現によって絞り込むことができる．以下，$w[i, j] = x\alpha y$ の接頭辞 x および接尾辞 y が α に隣接して現れるかを調べればよい．このことは，定理 1 を繰り返して用いることで，以下の処理に還元できる．

テキスト w とパターン P に対して $w[i, j] = P$ であることと以下は等価である．すなわちある変数の列 $A_1, A_2, \ldots, A_k$ が存在して，$u(A_1) \cdots u(A_k) = P$ であり，w の G による構文木 $T_G(w)$ において $A_1, A_2, \ldots, A_k$ が隣接して出現する．ここで，$u(A)$ は A を展開して得られる文字列であり，A, B が w の構文木 T において隣接するとは，$u(A)$ の最後の文字と $u(B)$ の最初の文字が w において隣接することを表す.

定理 1 より，各 A_i は十分長い文字列を符号化しているので，任意の P に対して，$k = O(\log |P|)$ となる．したがって，G から直接 $A_1, \ldots, A_k$ の隣接出現が判定できれば，テキスト中のパターン P の出現を $O(\log |P|)$ に近い時間で検出できる．

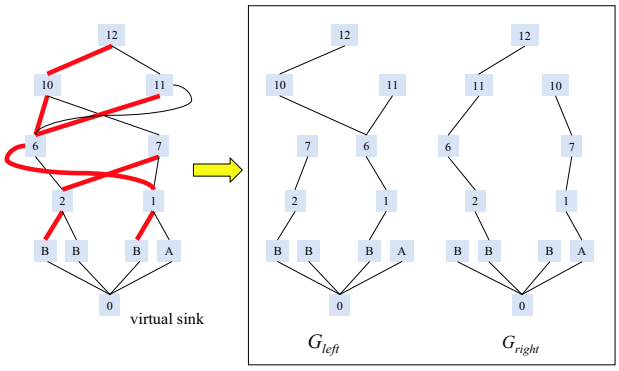
以下では，この隣接出現の検索のアイディアを図によって説明する．図 5 はあるテキストの構文木とその DAG 表現である．テキスト中に出現するパターン P を囲みによって表している．この P はアルゴリズムの性質により，P の出現によらず十分長い文字列を符号化した変数 6 を必ず含む．ここで，残りの接尾辞 BB を符号化した変数 2 が構文木中に隣接して出現すれば，P が出現したことがわかる．この検索を DAG 上で高速に行いたい.

この検索を実現するために，DAG G に唯一のシンクを付け加えたものを作り，この G を二つの部分グラフに分割する．一方は，G の各ノードの左の辺のみからなる G_{left} であり，他方は，右の辺のみからなる G_{right} である．圧縮アルゴリズムは常に digram を文字に置き換えるので，G の任意のノードはちょうど二つの子を持つ．したがって，G は必ず唯一の G_{left}, G_{right} に分割される．この分解の様子を図 6 に示す．ここで，以下の定理 2 が成り立つことに注意しよう.

[**定理 2**]　A と B が G 上でこの順で隣接するとき，またそのときに限り，A の最右先祖 X で，X の右の子 Y から左の辺のみをたどって B に到達できるものが存在する.

ここで A の最右先祖は以下のように定義される．G が表す構

図 6　DAG の分解



文木は 2 分木であるので，すべての枝は左右の順序を持ち，それぞれに属する枝を左辺および右辺と呼ぶ．そこで，構文木中のノード A から左辺をちょうど一回たどって到達できる A の先祖 X のうち，左辺を最後にたどるものを A の最右先祖と呼ぶ．直感的には，最右先祖とは自分の右側にある最も近い先祖である．

例えば図 5 において，ノード 2 の最右先祖はノード 7 や 10 がある．DAG の各ノードに対してこのような最右先祖 X を記憶することができる．また，その右の子 Y も DAG から直ちに参照できる．さらに Y から B へ左の辺のみをたどって到達できるかは，G_{left} で Y から X へ到達できるかを調べればよいが，これは木の最近共通祖先を計算できればよい．この計算は木に前処理を施すことで定数時間で判定可能である(注4)．以上により，A の右隣に B が隣接するかは定数時間で判定可能である．同様に最左先祖も定義され，ノード A の左の隣接関係も定数時間で判定可能となる．以上により，本提案手法に対して以下の定理を得る．

[定理 3]　テキスト w ($|w| = n$) にパターン P ($|P| = k$) が出現するか否かを，$O(k \log^* k + d \log k)$ 時間で判定可能である．このとき，テキストの前処理にかかる時間および領域はそれぞれ，$O(n \log^* n)$ および $O(g(\log^* n) \log n)$ である．ここで g は，圧縮アルゴリズムが生成した CFG に含まれる異なる変数の個数であり，d はその CFG を DAG で表現したときの最大次数である．

この定理における二つのパラメータ g, d は，テキストサイズ n に対して十分に小さいと考えてよい．実際，w の圧縮が容易であれば圧縮の最適解である g は十分に小さく，また w がランダムに近く圧縮ができない場合には，ほとんど置き換えが起こらないためやはり g は小さい．また d は CFG の構文木中のある変数の出現回数と考えられるが，これはパターンが長くなるにつれて急激に小さくなる．以上のことから，本研究で提案するデータ構造はテキストに前処理を施さない照合アルゴリズムよりも高速で，部分文字列に対する索引を構築する従来の手法よりも省スペースであると結論できる．

(注4)：最近共通祖先の定数時間判定アルゴリズムについては [4] などが詳しい．

4. ま と め

本研究では，テキスト w にパターン P が出現するか否かを判定する問題に対して，圧縮による前処理を施すことで，$O(d \log |P|)$ 時間で計算可能な索引構造を提案した．d は w に依存するが，w と比較して十分小さいと考えてよい．この問題は，通常のパターン検索問題よりも制限されている (出現位置を計算しない) が，データ全体を走査しないため，圧縮パターン照合による手法よりも高速に検索可能であり，また，本手法は，圧縮データそのものが索引となっているため，接尾辞木や接尾辞配列等の索引構造と比較して，使用する記憶領域が小さい．今後は，最大繰り返し等，検索できる対象を拡張することが目標である．

文　献

[1] A. Apostolico, and S. Lonardi. Some Theory and Practice of Greedy Off-Line Textual Substitution. *Proc. Data Compression Conference 1998*, 119-128.

[2] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Rasala, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inform. Theory*, 51(7):2554-2576, 2005.

[3] G. Cormode, and S. Muthukrishnan. The String Edit Distance Matching Problem With Moves. *ACM Trans. Algorithms*, 3(1), 2007.

[4] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Computer Science and Computational Biology. Cambridge University Press, 1997.

[5] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage System: a Unifying Framework for Compressed Pattern Matching. *Theor. Comput. Sci.*, 298(1):253-272, 2003.

[6] J. C. Kieffer, and E.-H. Yang. Grammar-Based Codes: a New Class of Universal Lossless Source Codes. *IEEE Trans. Inform. Theory*, 46(3):737–754, 2000.

[7] J. C. Kieffer, E.-H. Yang, G. Nelson, and P. Cosman. Universal Lossless Compression via Multilevel Pattern Matching. *IEEE Trans. Inform. Theory*, IT-46(5), 1227–1245, 2000.

[8] N.J. Larsson, and A. Moffat. Offline Dictionary-Based Compression. *Proceedings of the IEEE*, 88(11):1722-1732, 2000.

[9] E. Lehman, and A. Shelat. Approximation Algorithms for Grammar-Based Compression. *Proc. 20th Ann. ACM-SIAM Sympo. Discrete Algorithms*, 205-212, 2002.

[10] C. Nevill-Manning, and I. Witten. Identifying hierarchical structure in sequences: a linear-time algorithm. *J. Artificial Intelligence Research*, 7:67–82, 1997.

[11] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211-222, 2003.

[12] H. Sakamoto, S. Maruyama, T. Kida, and S. Shimozono. A Space-Saving Approximation Algorithm for Grammar-Based Compression. *IEICE Trans.*, to appear.

[13] H. Sakamoto, T. Kida, and S. Shimozono. A Space-Saving Linear-Time Algorithm for Grammar-Based Compression. *Proc. 11th International Symposium on String Processing and Information Retrieval*, LNCS3109:218-229, 2004.

[14] H. Sakamoto. A Fully Linear-Time Approximation Algorithm for Grammar-Based Compression. *J. Discrete Algorithms*, 3(2-4):416-430, 2005.

[15] J. Ziv, and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inform. Theory*, IT-23(3):337-349, 1977.

[16] J. Ziv, and A. Lempel. Compression of Individual Sequences via Variable-Rate Coding. *IEEE Trans. Inform. Theory*, 24(5):530-536, 1978.