データ圧縮の理論に基づく効率的な索引構造

An Effective Index Structure Based on a Data Compression Theory

多比良 嘉成*　岸上 直也†　田中 洋平‡　坂本 比呂志§

Kazunari TAIRA　Naoya KISHIUE
Youhei TANAKA　Hiroshi SAKAMOTO

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によって符号化するという特性を持つ．本研究では，この特性を応用して，符号化された文字とグラフラベリングによる補助情報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し，予備的な実験によってその有効性を示す．

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, and we also evaluate the performance of the algorithm by preliminary experiments.

1. はじめに

本研究では，圧縮されたデータに対する軽量・高速なパターン検索手法を提案する．ここで対象とする検索は，完全な部分文字列の一致を判定することであり，テキスト T とパターン P が与えられたとき，P が T に出現するか否かを判定する問題や P が T に出現する場所および回数を出力する問題のことである．したがって，正規表現やミスマッチを許した近似照合などは本研究の対象外である．

本研究で提案する圧縮法は，CFG 圧縮と呼ばれる枠組みに含まれる．CFG 圧縮では，あるテキストが与えられたとき，そのテキストのみを生成する制限された文脈自由文法 (CFG) を考え，そのような CFG のうちできるだけ小さなものを見つけることが目標である．例えば，開始記号 S から始まる CFG の規則 $\{S \to AA,\ A \to BB,\ B \to CC,\ C \to aa\}$ は，テキスト a^{16} のみを生成する最小の CFG である．ここで，テキストに出現する文字をアルファベットと呼び，この例における S, A, B, C のような置換文字を変数と呼ぶ．

もし，最適な CFG 圧縮が効率的に計算できたり，CFG を展開することなく元テキストに関する情報を取り出すことができれば，CFG を圧縮データ索引として利用できる．残念ながらこの最適化問題は NP-困難であるが，近似的解法についてはよい結果が知られている．

入力長を n とするとき，最適解に対する近似率 $O(\log n)$ を達成するいくつかのアルゴリズム [1, 12, 14] が知られており，圧縮率の理論的観点からは，これらはほぼ最適と見なしてよい．ただし，いずれも接尾辞木 [4] の構築などで $\Theta(n)$ 領域を必要とするため，これらの手法は規模耐性が低い．一方，理論的保証はないものの，実用的にはコンパクトな CFG を高速に求めることが可能であるため，LZ77 や LZ78 [17, 18] をはじめ，多くの圧縮法が提案されている [6, 7, 10, 15, 16]．

これらの CFG 圧縮は，最適性または規模耐性のいずれかの点で欠点がある．これに対して， [13] で提案された CFG 圧縮法は，アルファベットや変数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，$\log\log\cdots\log n > 1$ を満たす log の最大回数を表し，$n = 2^{65536}$ のような巨大な数に対しても $\log^* n = 5$ に過ぎないため，実用的には定数と考えてよい．したがって，これは準最適な圧縮と見なせる．そこで，この CFG からテキストに関する情報をすばやく取り出すことができれば，省スペースかつ高速な索引として利用できる．

これまでの圧縮法が，テキスト中の部分文字列に対する何らかの索引を必要としていたのに対して，[13] 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 CFG の変数以外の領域を必要としないため，飛躍的に省スペースであることが保証できる．本研究で提案する圧縮データ索引は，この圧縮アルゴリズムに基づいている．

[13] の CFG 圧縮では，アルファベット還元 [3] と呼ばれる文字の変換を利用している．ここでその概略を述べる．テキストの k 番目の文字とその直前の文字との単純な演算 (例えば加算) によってある文字を決める．この文字をテキストの k 番目の文字のラベルと呼ぶ．k を動かすことによって，テキストと同じ長さのラベルの列ができる．ここで重要なことは，テキストに文字 A が複数出現するとき，それらの左隣の文字が同じならば，それらの A のラベルは必ず一致するということである．同様にこのラベル列をテキストと見なして同じ計算を行うと，今度は長さ 2 の文字列が一致するという情報を次のラベルに埋め込むことができる．この計算を繰り返すことで，テキスト中の任意の長さの部分文字列に関する情報をラベルに埋め込むことができる．この計算は隣り合う文字同士の単純な演算にすぎないため，補助的なデータ構造や部分文字列に関する情報を一切必要としない．[13] の CFG 圧縮では，この埋め込まれた情報を利用することで効率的に同一部分文字列の置換が可能になる．

圧縮データからパターンを検索しようとするとき，テキストの中身が何らかの意味でシャッフルされている場合，原テキストを部分的に復号する必要がある．本提案手法に用いる圧縮法は，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．この性質はより正確には，テキスト中のパターン P の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．ここで，A はある一文字の変数であり，x, y, u, v はパターン P に対して十分に短い文字列を表す．

この関係は，テキスト中にパターン P が出現するための必要十分条件であるため，この性質を用いて圧縮データに対する索引が構築できる．その基本アイディアは，以下のように説明できる．テキスト中にパターン P が出現するには，辞書に変数 A が含まれることが必要であり，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 (q を十分に小さい自然数としたとき，P のおよそ $1/q$ に相当する長さ) を符号化していることが保証できるため，残りの接尾辞・接頭辞に対して上記の再帰的なパターン検索を行うことで，P の長さの対

*九州工業大学大学院情報工学府
taira@donald.ai.kyutech.ac.jp
†九州工業大学大学院情報工学府
n_kishiue@donald.ai.kyutech.ac.jp
‡凸版印刷株式会社西日本事業本部
yohei_1.tanaka@toppan.co.jp
§正会員 九州工業大学大学院情報工学研究院および科学技術振興機構
hiroshi@ai.kyutech.ac.jp

数に相当する時間でこれらの変数のチェックを行うことができる．

実際の検索時間は，これに圧縮された CFG の複雑さに関連する係数が掛かるが，それは原テキストに対して十分小さく，その解析については本論の中で説明する．結論として，本手法により，全体をテキスト長にほぼ無関係な時間で行うことができる．本研究ではこの処理を高速に実行するための索引構造を提案し，アルゴリズムによって置換された各変数がどの程度のパターンを復号しているかを予備的な実験によって測定することで本研究の有効性を確かめる．

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 [3] 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

テキストに最初から含まれる文字をアルファベットと呼び，それらの集合を Σ で表す．それらの文字とは別に，アルゴリズムが置換のために生成する文字は変数と呼んで区別する．今，アルファベットと変数を含む文字の集合に対して，それらの文字すべてを葉として持つ完全平衡 2 分木を考え，それをアルファベット木と呼ぶ．図 1 にサイズ 11 の場合のアルファベット木を示す．置換による新しい変数が生まれるたびにアルファベット木の葉が増えていくが，ここでは簡単のため，あらかじめ十分に大きなサイズの木を固定しておくことにする．また，葉に割り当てる変数の順番は任意でよい．

この木によって，任意の文字の組 (a_i, a_j) に対して，$lca(i,j)$ を文字 a_i と a_j のアルファベット木における最近共通祖先の高さとして定める．例えば図 1 によると，(a_2, a_4) に対しては，$lca(2,4) = 2$ である．以下，簡単のために，文字 a_i と整数 i を同一視する．これを用いて，$w[k]$ に対するアルファベット還元 $R(w[k])$ を以下のように定義する．ここで，$w[k]$ はテキスト w の k 番目の文字を表し，$w[i,j]$ は w の i 番目の文字から j 番目の文字までの部分文字列を表す．

$k = 2, 3, \ldots, |w|$ であるときは，$w = [k-1, k] = a_i a_j$ に対して $R(w[k]) = 2lca(i,j) \ \ (i < j)$ または $R(w[k]) = 2lca(i,j) + 1 \ \ (i > j)$ と定め，$k = 1$ であるときは，$w = [1,2] = a_i a_j$ に対して $R(w[1]) = 2lca(i,j) \ \ (i > j)$ または $R(w[1]) = 2lca(i,j)+1 \ \ (i < j)$ と定める．

この定義は，$i = j$ の場合，すなわち同じ文字が連続する場合については未定義であるが，ここではまだそのような一般の文字列は考えなくてよい．この関数 $lca(i,j)$ によって，文字の集合 $\{1, \ldots, n\}$ から構成される同じ文字が連続しない文字列が，その部分集合 $\{1, \ldots, \lceil \log n \rceil + 1\}$ から構成される元の文字列と同じ長さの整数列に還元される．このとき，還元後の文字列 (整数列) も同じ文字が決して連続しないことが容易にわかる．したがって，この操作は再帰的に実行可能である．この変換をアルファベット還元と呼ぶ．

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．

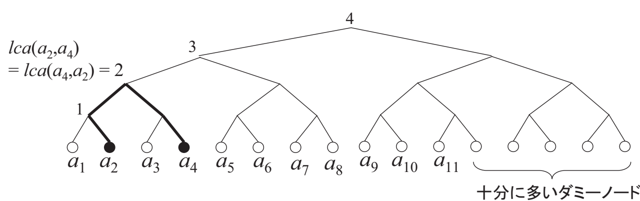

図 1. アルファベット木

Fig. 1. Alphabet Tree

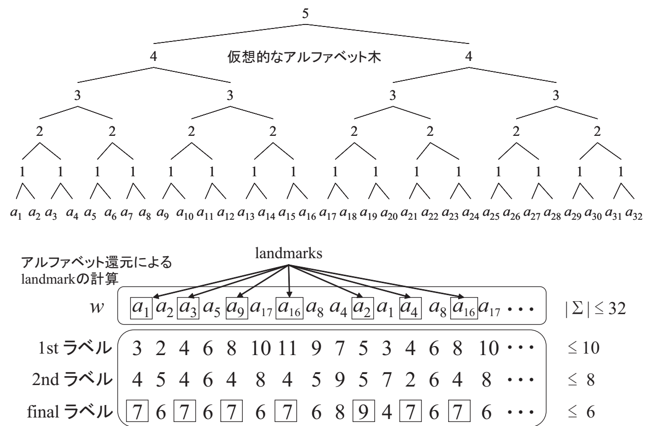

図 2. アルファベット還元

Fig. 2. Alphabet Reduction

以降では，テキストの圧縮法を説明する．この圧縮法では，与えられた文字列を 2 種類の部分文字列に分解し，$ababcba \cdots$ のように同じ文字が連続しない部分については，前述のアルファベット還元を適用して圧縮し，$aa \cdots a$ のように同じ文字が連続する部分については，左端から 2 文字ずつをある変数に置き換えていく．

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 $w = w_1 w_2 \cdots w_m$ 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(長さ 2 の部分文字列) を一文字で置換して，$A \to aa$ 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，その左隣の文字とで構成される digram を適当な変数によって置換する．ただし，その landmark が最左文字ならばこの操作はスキップする．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 を左側優先で適当な変数で置換していく．ここで適当な変数とは，現在置換しようとしている digram の置換すべき変数がすでに辞書に登録されていれば，常にそれに従うということを意味する．

[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．

このアルゴリズムによって文字列が実際にどのように圧縮されているかを図 4 に示す．この図で示されている文字列は，$f_0 =$ 'a'，$f_1 =$ 'b'，$f_{n+1} = f_{n-1} \cdot f_n$ で定義される**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．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．図 4 に示されている構文木の内部頂点が各 digram を置換した変数を表している．この例から，同じ部分文字列がなるべ

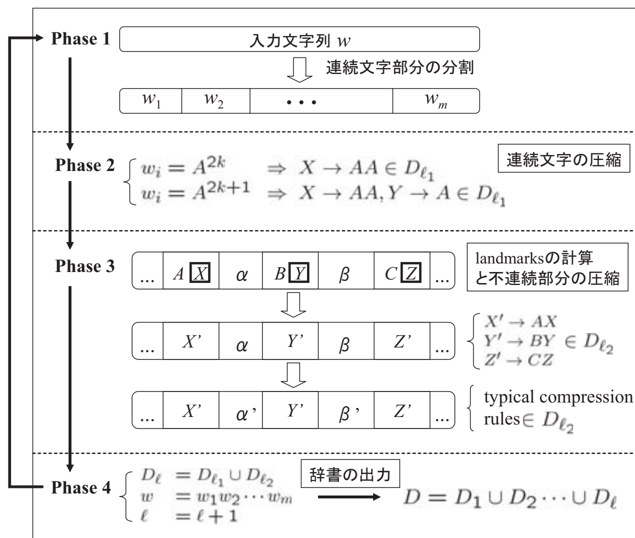

図 3. 圧縮アルゴリズムの概略

Fig. 3. Outline of Compression Algorithm

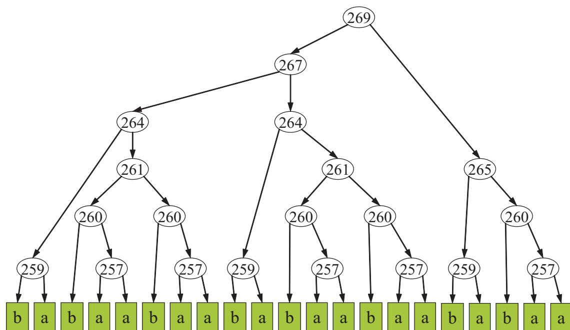

図 4. フィボナッチ文字列 (部分) の圧縮結果

Fig. 4. Compression Result for Fibonacci String

く同じ変数で置換されている様子がわかる．

文献 [13] で示されているように，あるテキスト w に現れる部分文字列の異なる出現 $w[i,j] = w[k,\ell] = \alpha\beta\gamma$ に対して，$w[i,j]$ と $w[k,\ell]$ のランドマークの位置は β の部分は完全に一致し，$|\alpha\gamma| = O(\log^* n)$ である．ランドマークが一致している部分は完全に同じ変数への置き換えが行われ，n を十分に大きくしても $\log^* n < 6$ と見なしてよい．このことは，圧縮アルゴリズムの毎ループごとに成り立つので，ある有理数 $0 < q < 1$ が存在して，$w[i,j]$ と $w[k,\ell]$ の構文木における最大共通部分木の大きさは，少なくとも $q(j-i)$ 以上であると保証できる．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圧縮データ索引を提案する．

3. CFG による圧縮データ索引

本節では，CFG 圧縮を用いた圧縮データ索引と検索手法について述べる．この基本アイディアは，テキスト w の圧縮辞書によってパターン P を同様に圧縮し，その変数列を w の圧縮データの中に順序を保って埋め込めるか否かによって，P の出現を決定することにある．

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，$w[i,j]$ 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 $(i < j)$．

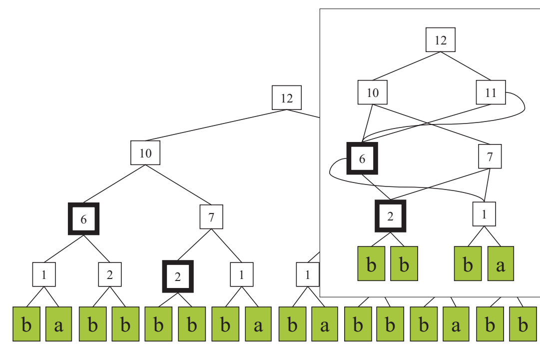

図 5. 構文木と DAG

Fig. 5. Parsing Tree and DAG

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．

この定理は，w に含まれる隣り合う landmark は常に 6 文字以内に接近していることおよび $w[k]$ が landmark であるか否かはその左側の高々$\log^* n (\sim 5)$ 文字以内の文字だけで決定されることから容易に導かれる．

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 $\log^* n$ 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 x, y が十分に短い文字列であれば，それを展開して検索すればよく，そうでないならば，それらの文字列に対して定理 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アルファベット文字列であり，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 は圧縮の特性により，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 がすぐ隣に出現すれば P が出現したことがわかる．

この問題は，任意の変数 A, B を根とする部分木が構文木内で隣接するか否かを解く問題に帰着できる．まず，A の最右先祖を以下のように定義する．構文木は 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．ここで，A の先祖で左辺をちょうど一回たどって到達できるノードのうち A に最も近いノードを 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．

この最右先祖の概念を用いて，ノードの隣接関係を以下のように簡潔に表すことができる．

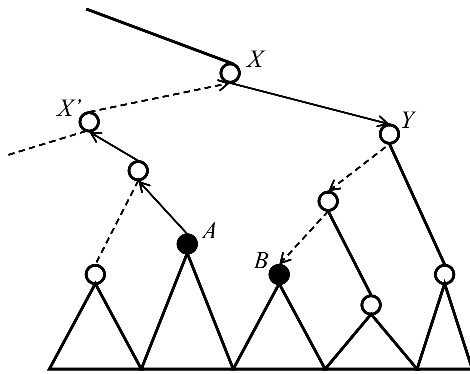

図 6. 部分木の隣接関係

Fig. 6. Adjacent relation of subtrees

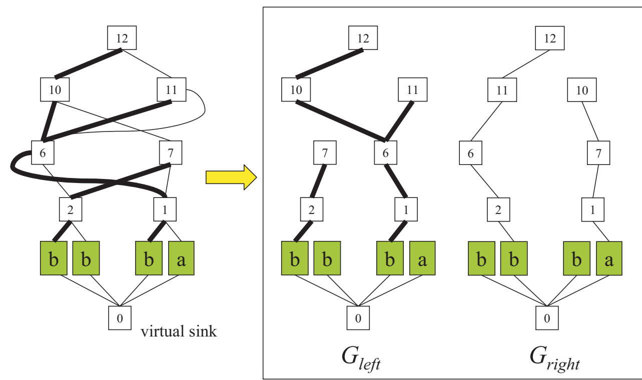

図 7. DAG の分解

Fig. 7. Decomposition of DAG

定理 2 A, B を根とする部分木が，構文木内でこの順番に隣接することと以下を満たすある変数 X とその右の子供 Y が存在することは等価である．

1. X は A の最右先祖
2. Y から左辺だけをたどって B に到達可能

この関係を一般的に表したものが図 6 である．A, B が隣接するためには図のような矢印で表された経路が存在すればよい．構文木中の任意のノードに対して，その最右先祖は唯一であるので，それを前処理して覚えておくことで，構文木が与えられた場合の A, B の隣接関係の判定は定数時間で可能である．

しかしながら，本研究で目指す圧縮索引では，構文木は実際には図 5 中の右側に描かれている DAG のように，いくつかのノードが縮約されているため，ある変数 A の最右先祖は唯一ではない．DAG 構造においても隣接関係を簡単に判定するために，得られた DAG を二つに分解することを考える．まず，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 の各ノードの左辺のみからなる G_{left} であり，他方は，右辺のみからなる 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 7 に示す．

DAG 上でも図 6 のような隣接関係を判定するために，まず，DAG における変数 A の最右先祖 X のひとつが確定したと仮定する．G_{left}, G_{right} は一番下のノードが根であることに注意すると，構文木における X の右の子供 Y は G_{right} における X の親であり，構文木において Y から B へ左辺のみをたどって到達可能であることと G_{left} において B が Y の先祖であることは等価である．木構造の先祖子孫関係は定数時間で計算可能¶であるので，結局この問題は，DAG 上で A の最右先祖 X をどのように見つけるかという問題に帰着できる．

構文木において，A からその最右先祖 X に到達する直前には必ず左辺をたどっており，それ以外ではすべて右辺をたどっているので，このようなノード X 全体は，G_{right} における A の子孫を X' とすると G_{left} において X' の子供すべてと同じである．このような処理は木構造を一度だけ巡回すれば計算可能である．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予想できる．次節ではこの見積もりを予備的な実験によって検証する．

4. 実験

本研究の実験は，Pentium Dual-Core CPU (2.70GHz)，4GB メモリ上の Windows Vista において行った．

図 8 は，前々節で定義されたフィボナッチ文字列の圧縮率を gzip と比較した結果である．横軸のパラメータは定義の漸化式の反復回数を表しており，この値が大きくなるにつれてデータサイズは指数的に増加することがわかる．

本研究で採用したアルファベット還元法による CFG 圧縮は，ほぼ最適な圧縮であるのに対して，gzip では一定以上の入力サイズ以降は圧縮率が急激に悪化していることがわかる．これは，gzip が高速に実行できるように，履歴情報を格納する領域サイズをあらかじめ固定しているためである．この領域を入力の数倍程度確保すれば CFG 圧縮と同等の圧縮率を達成できる可能性があるが，それでも gzip 等の従来の圧縮法は，前節で示した CFG 圧縮が持つ構造を保存する特性がないため，索引として利用するためにはデータを展開したり補助情報の追加が必要である．したがって，この意味でも本研究の手法が優れている．

次に，前節で示したパターン検索の時間を見積もるための予備的実験結果を示す．テキスト w を圧縮した結果を DAG G_w とする．パターン P がテキスト w に出現することは，P を w の辞書で圧縮したときに決定される極大な変数列 $A_1, A_2 \ldots, A_k$ が G_w に隣接関係の順序を保って埋め込み可能であることと等価である．二つの変数のグラフ内における隣接関係は定数時間で判定可能であるので，パターン検索の時間は k の大きさと各 A_i の出現回数に依存する．

まず k のサイズを見積もる．図 3 のアルゴリズムのループ 1 回

¶木構造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．

あたりの処理では，landmark のズレはパターン P の左右の定数(高々5 文字) 以内に収まる．このことから，w に P が出現するならば，G_w はある変数 A を含み P は xAy のように圧縮され，A は P の少なくとも $1/q$ 以上の部分文字列を符号化している．ここで，q は w, P とは無関係の十分小さい自然数である．この q は具体的に見積もることができるが，本論文では詳細は省略する．

このように w, P に対して定まる最長の文字列を符号化している変数 A をパターン P のコアと呼ぶ．同様に，$A_1, A_2 \ldots, A_k$ は P の接頭辞・接尾辞に対して定まるコアの列である．有理数 $1/q$ は入力とは無関係の定数であるので，$k = O(\log |P|)$ である．

次に k の実測値がパターン長の対数に近いことを圧縮の実験結果によって示す．前述のように，アルゴリズムによる landmark のズレは，パターンの定数長の接尾辞・接頭辞に限られる．したがって，圧縮の構文木における任意の変数が，その高さの指数に比例した文字列を符号化しているならば，コアが符号化している P 文字列が十分に長く結果として $k = O(\log |P|)$ であるといえる．

以上のことを，圧縮の実験結果から得られた構文木を解析することで裏付ける．図 9 は，各コアが符号化している部分文字列長とそのコアの構文木中の高さの関係を示したものである．実験は，フィボナッチ文字列と自然言語テキスト (ロイターニュース記事) に対して行った．テキストのアルファベットサイズに違いがあるためそれぞれのデータで傾きが異なるが，いずれの場合もほぼ直線と見なせる．これは，構文木中の高さ h における変数 (コア) が，平均 $\alpha 2^h$ の長さの部分文字列を符号化していることを意味する．ここで，α はテキスト中のアルファベットに依存する定数である．この結果から，実用的にも $k = O(\log |P|)$ であることが実験によって裏付けられた．

最後に，提案手法による検索性能を他手法と比較した結果を示す．ただし，現時点では前節の定理 2 によってパターンの出現をコアの隣接関係のみで判定するアルゴリズムを実装できていない．そこで本実験では，パターンのコアをひとつ確定してその周辺部分を展開して KMP オートマトンで照合することによってパターンの出現を確定している．図 10 がその結果である．実験に使用したデータは， [5] で公開されているイネゲノムの塩基配列を繋げた約 50MB のテキストである．このテキストからランダムに切り出した，長さ 100〜30000 の比較的長い部分文字列をパターンに指定し，パターンの出現を判定するために掛かった時間を測定した．比較対象は，短いパターンに対しては最高速の圧縮文字列照合である KMPonBPEX 法 [9] とパターンが長くなると平均して高速になる BMH 法 [4] である．BMH 法に符号化を組み込んだ手法 [11] も存在するが，今回の実験のようにパターンが十分に長い場合には BMH 法と差がなくなるため今回は割愛する．

図 10 のデータから，パターン長が 1KB 以降で提案手法の検索時間が徐々に増加しており，パターン長が 10KB を超えるあたりで BMH 法に逆転されていることがわかる．これは，本来は必要のない余分な展開と照合を繰り返すことによるオーバーヘッドが大きくなりすぎるためである．実際，それらの時間がほとんど掛からない短いパターンに対しては他手法よりも高速であることがわかる．したがって，変数の隣接関係のみでパターンを検知する改良を加えることで，より広範囲な優位性を期待できる．

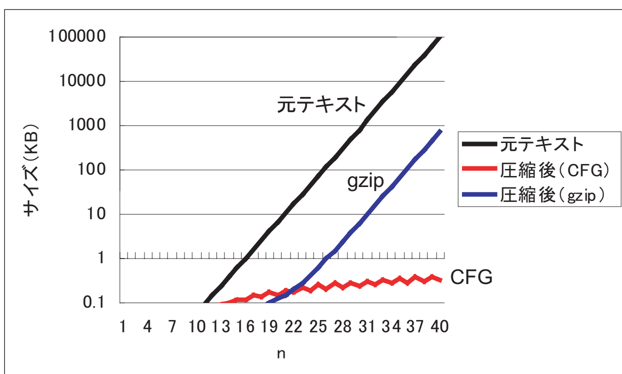

図 8. 圧縮率の比較 (対 gzip)

Fig. 8. Comparison of Compression Ratio v.s. gzip

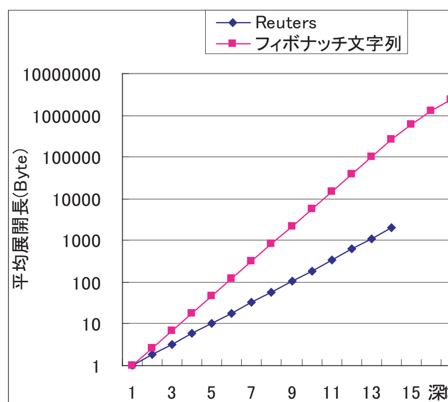

図 9. コアの平均展開長

Fig. 9. Average of Encoded String Length by Core

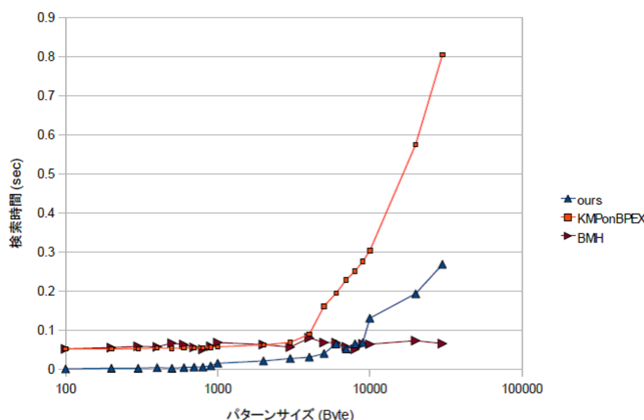

図 10. 検索時間の比較

Fig. 10. Search time comparison

5. おわりに

本研究では，データをあらかじめ圧縮することで検索を効率的に行う枠組みに対して一手法を提案した．圧縮に基づく手法は，Web データのように複雑な編集が頻繁に行われるデータに対しては不向きであるが，特許データや遺伝子データのようにほぼ追加や削除のみが行われる静的なデータに対しては類似部分の発見等の応用が見込まれる．

また，パターンが数キロバイト程度の長いものになると，コアが符号化する部分文字列もそれに比例して長くなるため，パターンの出現とそのコア 1 個の出現がほぼ一致する．なぜならば，十分に長いパターンが 2 回以上出現する可能性は極めて低く，同様にそのコアも十分な長さを持っているからである．このコアの長さは，前節の実験データによるとパターン長の約 10 %になっている．したがって，本手法はある Web ページや特許 1 件分の文書

など比較的長いパターンの検索に対して高速であり，この性能はテキストサイズに対して頑健である．

提案手法では，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，$O(d \log |P|)$ 時間で計算可能な圧縮データ索引を示した．現在，理論的に最も効率がよい圧縮索引は，文献 [2] で提案されている SLP に基づく圧縮索引である．SLP とは，本研究と同様に $Z \to XY$ の形に制限された規則のみからなる CFG のある部分クラスである．この圧縮データとパトリシアトライなどのデータ構造を組み合わせることで，この手法では $O((k^2+occ)h \log g)$ 時間で検索を行うことが可能としている．ここで，occ はパターン P の出現回数，h は SLP の構文木の高さを表す．仮に $occ = 1$ とした場合，検索時間は $O(k^2 h \log g)$ となるが，[2] でも述べられているように，テキストサイズが巨大になると $\log g$ の影響が大きくなり実用的な手法でなくなる．実際，このアルゴリズムは実装されていない．一方，本研究で提案した手法の検索時間は，パターンが存在するか否かの単純なものに限るが，$O(k \log^* k + d \log k)$ 時間で実行可能である．このうち最も影響が大きい項は d であるが，この値はパターン長の増加と共に急激に小さくなるので，本研究はパターンが十分に長い問題に対しては他手法と比較して優位性があると考えられる．またこの見積もりは，実際にアルゴリズムを実装して圧縮文字列照合法と比較することによっても裏付けることができた．また，d は P の出現回数と関連するパラメータであるが，コアの平均展開長の測定実験から，P がある程度の長さを持つ場合には d は w と比較して十分小さいと考えてよい．したがって，検索のためにデータ全体を走査しないため，従来の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従来の索引構造と比較して要求される記憶領域が小さい．これらのことは，ベンチマーク等の予備実験からある程度確認できたが，今後は完全なパターン検索を実装し，他の検索手法との性能比較を行うことで，本研究の優位性を示したい．

[謝辞]

本研究は，JST 戦略的創造研究推進事業 (さきがけ) および JST 地域イノベーション創出総合支援事業 (シーズ発掘試験) の支援を受けた．また，適切なコメントを頂いた皆様に感謝します．

[文献]

[1] 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.

[2] F. Claude and G. Navarro. Self-indexed Text Compression Using Straight-Line Programs. *Proc. MFCS 2009*, 235-246

[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] The International Rice Genome Sequencing Project (IRGSP), `http://rgp.dna.affrc.go.jp/E/IRGSP/`

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

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

[9] S. Maruyama, Y. Tanaka, H. Sakamoto, M. Takeda. Context-Sensitive Grammar Transform: Compression and Pattern Matching. *IEICE Trans. on Information and Systems*, to appear.

[10] C. Nevill-Manning and I. Witten. Compression and Explanation Using Hierarchical Grammars. *Computer Journal*, 40(2/3):103–116, 1997.

[11] J. Rautio, J. Tanninen, and J. Tarhio. String Matching with Stopper Encoding and Code Splitting. *Proc. CPM2002*, 42–52, 2002.

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

[13] H. Sakamoto, S. Maruyama, T. Kida, and S. Shimozono. A Space-Saving Approximation Algorithm for Grammar-Based Compression. *IEICE Trans.*, Vol.E92-D,No.2,pp.158-165, 2009.

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

[15] T.A. Welch. A Technique for High Performance Data Compression. *IEEE Comput.*, 17:8-19, 1984.

[16] E.-H. Yang, and J.C. Kieffer. Efficient Universal Lossless Data Compression Algorithms Based on a Greedy Sequential Grammar Transform–Part One: without Context Models. *IEEE Trans. Inform. Theory*, 46(3):755-777, 2000.

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

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

多比良 嘉成　Kazunari TAIRA

2008 年 3 月 九州工業大学情報工学部知能情報工学科卒業．現在，同大学情報工学府情報科学専攻に在学中．データ圧縮の理論を応用した高速情報検索の研究に従事．

岸上 直也　Naoya KISHIUE

2009 年 3 月 九州工業大学情報工学部知能情報工学科卒業．現在，同大学情報工学府情報科学専攻に在学中．データ圧縮の理論に関する研究に従事．

田中 洋平　Youhei TANAKA

2007 年 3 月 九州工業大学情報工学部知能情報工学科卒業．2009 年 3 月 同大学大学院情報工学府情報科学専攻修士課程修了．在学中は，データ圧縮とその応用に関する研究に従事．現在，凸版印刷株式会社 西日本事業本部勤務．

坂本 比呂志　Hiroshi SAKAMOTO

九州工業大学大学院情報工学研究院准教授および科学技術振興機構さきがけ「知の創生と情報社会」研究者 (兼任)．1998 年九州大学大学院システム情報科学研究科情報理学専攻博士後期課程修了 (日本学術振興会特別研究員)．博士 (理学)．1999 年から 2003 年 7 月まで九州大学大学院システム情報科学研究科助手．機械学習，半構造データからの知識獲得および半構造データに対する索引付けに関する研究に従事．2006 年度人工知能学会論文賞などを受賞．電子情報通信学会，人工知能学会，日本データベース学会各会員．