HOKKAIDO UNIVERSITY

Title	STVF符号 ： 頻度刈り込み接尾辞木を用いた効率よいVF符号化
Author(s)	喜田, 拓也
Citation	日本データベース学会論文誌, 8(1), 125-130
Issue Date	2009-06
Doc URL	https://hdl.handle.net/2115/47126
Type	journal article
File Information	bsj-journal-08-01-125.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

DBSJ Journal, Vol.8, No.1
June 2009

STVF 符号：頻度刈り込み接尾辞木を用いた効率良い VF 符号化

STVF Code: An Efficient VF Coding using Frequency-base-pruned Suffix Tree

喜田 拓也♥
Takuya KIDA

本論文では，刈り込み接尾辞木を用いた新たな可変情報源系列固定長符号化（VF 符号化）の手法を提案する．この符号化手法は，頻度情報に基づいて刈り込んだ接尾辞木を文節木として用いて VF 符号化する．VF 符号は，すべての符号語が等長であるという工学的に好ましい性質があり，圧縮パターン照合などへの重要な応用がある．実験の結果，提案符号は，自然言語文書などに対して約 41%の圧縮率を達成しており，良く知られている Huffman 符号や，古典的な VF 符号である Tunstall 符号よりも圧縮性能が良いことがわかった．

In this paper, we propose a new VF-coding method. It uses a frequency-base-pruned suffix tree as a parse tree. VF codes have some desirable features from engineering aspects, and there are some important applications such as compressed pattern matching. Experimental results show that the proposed code achieves the compression ratio of about 41% for a natural language text, which is better than Tunstall code and Huffman code.

1. はじめに

本論文では，圧縮上でのパターン照合に適したデータ圧縮手法として，可変長情報源系列固定長符号化（VF符号化）を改善する手法（STVF符号化）を提案し，その性能について考察する．

ここでいうパターン照合とは，テキストとパターンと呼ばれる二つの文字列が与えられたときに，テキスト中に出現するパターンを見つけだす問題のことであり，文字列照合問題とも呼ばれる．代表的なアルゴリズムとしては，KMP法やBM法などが挙げられる[1]．一方のデータ圧縮は，データ中に含まれる冗長性をコンパクトに表現することで，記憶のための領域を削減する技術である．Huffman符号やLZ系圧縮法，BW変換に基づく手法など，これまで数多くの圧縮手法が提案されており，今なお盛んに研究されている[2][3]．

圧縮上でのパターン照合とは，データ圧縮されたテキストに対して，それを復元することなくパターン照合を行うことである．これは圧縮照合問題(Compressed matching problem)と呼ばれており，90年代初頭[4]で提案され，以降盛んに研究が行われている．当初この問題は，主に理論的な興味から始まったが，大規模テキストデータベースが個人で比較的簡単に取り扱えるようになった今日では，実用上の重要な課題となってきている．

圧縮照合問題は，形式的には，テキスト$T = t_1 \ldots t_u$が圧縮された形$Z = z_1 \ldots z_n$で存在し，パターン$P = p_1 \ldots p_m$が与えられたとき，T中のPの出現を，PとZのみを用いて見つけだすことと定義される．単純な方法としては，まずZをTに復元してから通常用いられるパターン照合アルゴリズムを用いる方法がある．この場合，照合に$O(m + u)$時間かかることに加え，復元のための時間もかかる．圧縮照合問題に対する最適なアルゴリズムとは，最悪時$O(m + n + R)$時間でパターン照合を行うアルゴリズムである．ここで，Rはパターンの出現回数である．しかしながら，テキストを効率良く圧縮することとパターン照合を高速に行うことを両立するのは容易ではない．

90年代後半から2000年初頭に入り，実際的に高速に圧縮照合を可能とする手法が出現した[5][6]．それらは，圧縮データに対して，データを復元した後に照合する方法よりも高速であるばかりでなく，元のテキストに対して照合するよりもおよそ圧縮率程度に照合を高速化することができる．その鍵は，圧縮率を犠牲にしてでも照合に適した圧縮法を選択する点にある．そのような圧縮法は，共通して次のような性質を備えている．

(a) 固定長符号である．特に符号長がバイトの整数倍であることが望ましい．

(b) 静的でコンパクトな辞書を用いる圧縮法である．

これらの性質は，圧縮率という観点からは大きな制約となる．しかし，さらなる照合速度の向上を達成するためには，このような性質を満たしつつ，より圧縮率の高い符号化手法を開発することが要求される．

良く知られたHuffman符号のように，テキストの固定長の部分文字列に可変長の符号語を割り当てる圧縮法は，FV符号(Fixed-length-to-Variable-length code)と呼ばれる．（静的な）Huffman符号は，静的でコンパクトな辞書を用いた圧縮法であるが，各文字が可変長の符号語に変換されるため，符号語の区切りが圧縮データ上では判別しにくく，それゆえ，圧縮上でのパターン照合にはやや不利な面がある．対照的に，VF符号(Variable-length-to-Fixed-length code)は，テキストの可変長な部分文字列に対して固定長の符号語を割り当てることで圧縮を行う圧縮法であり，符号語の区切りは明確である．

VF符号は，文節木と呼ばれる木構造を用いてテキストを可変長のブロックに分割し，それぞれのブロックに固定長の符号語を割り当てる．Huffman符号で用いられるHuffman木は，葉に情報源の記号が割り当てられ，各辺には符号語の記号が割り当てられるが，VF符号で用いられる文節木は，各辺に情報源記号が割り当てられ，葉に符号語が割り当てられる．

代表的なVF符号であるTunstall符号[7]は，非常に古くから知られているにもかかわらず，Huffman符号と比べて注目される機会が少なく，実応用で用いられることも皆無であった．しかし，すべての符号語が等長であることに加え，静的で比較的コンパクトな辞書を用いた圧縮法であるため，圧縮照合問題には非常に適した圧縮法であるといえる．さらに，Tunstall符号はHuffman符号同様，記憶のない情報源に対してエントロピー符号であることが証明されており，極限では情報源のエントロピーにまで平均符号長が漸近する．しかし

♥ 正会員　北海道大学大学院情報科学研究科　kida@ist.hokudai.ac.jp

ながら，実際のテキストに対してどの程度の圧縮率を有するのか，実証実験を報告した論文は皆無であり，ほとんど知られていない．

本論文では，Tunstall符号の実際の圧縮率を報告するとともに，VF符号の枠組みで圧縮率を改善する新しい符号化手法を提案する．提案VF符号(STVF符号)は，文字列の頻度による刈り込みを施した接尾辞木(Suffix tree; [1]を参照)を文節木として用いる．接尾辞木は，与えられたテキストのすべての部分文字列を格納するデータ構造であり，任意の部分文字列の頻度を接尾辞木上であらかじめ計算しておくことができる．圧縮照合問題では，パターン照合の対象となるテキストが既に与えられていることを仮定してよいので，圧縮対象のテキストに対する接尾辞木は最適な文節木の土台となりうる．今回，Tunstall符号化およびSTVF符号化を実際に実装し，その圧縮性能の評価実験を行った結果を示す．また，圧縮上でのパターン照合に関する速度比較実験についても報告する．

なお，本論文は，[15][16]を元に加筆・修正を行ったものである．

2. 関連研究

1990年にZivは，マルコフ情報源VF符号がFV符号よりも早くエントロピーに漸近することを証明した[8]．TjalkensとWillemsもまた，マルコフ情報源に対するVF符号の研究に取り組み，実用的な符号・復号化の実装方法を示している[9]．Savariは，記憶のある情報源に対するTunstall符号の効率について詳細な解析を行っている[10]．1997年までのFV符号およびVF符号については，網羅的な調査がAbrahamsによってなされている[11]．また，Visweswariahらは，辞書を用いないVF符号の性能について報告している[12]．

YamamotoとYokooらはVF符号の興味深い改善手法について提案している[13]．VF符号において各符号語は文節木の葉に割り当てられるが，彼らのアイデアでは，文節木の内部接点にもコードを割り当てることで無駄な枝を刈り込んでいる．そのような符号語は，一見，情報源系列上での語頭条件を満たさないように見えるが，VF符号においてはすべての符号語が等しい長さであるため，問題なく復号化できる．

圧縮照合問題に関して，良い性能を達成するアルゴリズムが既にいくつか提案されているが，特にバイトペア符号化法(BPE法)[5]やStopper Encoding法(SE法)[6]上の圧縮照合アルゴリズムは，元のテキストに対して照合を行うよりも高速に照合できることが知られている．ただしこれらの圧縮法は，Huffman符号と比べても圧縮率が低い．

ごく最近，MaruyamaらによってBPE法の圧縮率を大幅に改善する手法が提案された[14]．BPE法は，一種の文脈自由文法に基づく圧縮法で，頻出する文字ペアをテキストで使用されていない記号で置き換える操作を繰り返すことで圧縮する．Maruyamaらの改善策は，BPE法を文脈依存文法に基づく手法へと拡大し，256個に制限されていた辞書サイズを仮想的に拡大した点にある．

本論文で提案するSTVF符号と，まったく同じアイデアに基づく手法が，ほぼ同時期にKleinとShapiraらによって提案されており，同様の結果が報告されている[17]．ただし，彼らの手法では，接尾辞木の刈り込みが木の深さ優先順の探索に基づいており，本論文で提案している幅優先順的な刈り込みとはアルゴリズムが異なっている．

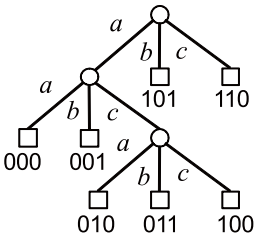

図 1. 文節木の例

3. 準備

3.1 記法と用語の定義

Σを有限アルファベットとする．Σ*はΣ上の文字列すべてからなる集合である．文字列$x \in \Sigma^*$の長さを|x|と書く．長さが0の文字列を空語と呼び，εで表す．したがって，$|\varepsilon| = 0$である．また，$\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$と定義する．二つの文字列$x_1$と$x_2$を連結した文字列を$x_1 \cdot x_2$で表す．特に混乱がない場合は，これを$x_1x_2$と略記する．

文字列x，y，zは，$w = zyz$であるとき，それぞれwの接頭辞，部分文字列，接尾辞と呼ばれる．文字列wのi番目の記号をw[i]で表す．また，wのi番目から始まりj番目で終わる部分文字列をw[i:j]と書く．ただし，簡便のため，$j < i$のとき$w[i:j] = \varepsilon$とする．また，文字列$w \in \Sigma^*$のすべての部分文字列からなる集合をFac(w)と書く．

3.2 Tunstall 符号

$\Sigma = \{a_1, \ldots, a_k\}$を大きさ$k \geq 1$の情報源アルファベットとする．また，Σの各要素は，添字の順で順序付けされているものとする．

任意の整数$m \geq 1$について，内部接点の数がm個で，かつ，各辺にΣの要素がラベル付けされた順序付きk分木を文節木$\mathcal{T}_m$とする．また，この木の大きさをmと定義する．以下ではこの$\mathcal{T}_m$を用いてテキストを VF 符号化することを考える．

まず，$\mathcal{T}_m$中のすべての葉に$\lceil \log N \rceil$ビットの整数で番号付けを行う．ここで，Nは$\mathcal{T}_m$の葉の個数である．このとき，$\mathcal{T}_m$によるテキストの符号化は以下の手順で行われる．

1. $\mathcal{T}_m$の根を探索のスタート地点とする．
2. 入力テキストから記号を1個読み取り，文節木$\mathcal{T}_m$上の現節点からその記号でラベル付けされた子へと移る．もし，葉に到達したら，その葉の番号を符号語として出力し，探索の地点を根へ戻す．
3. ステップ 2 をテキストの終端まで繰り返す．

図 1 のような文節木で，テキストT = aaabbacb を符号化すると，符号語の系列は 000 001 101 011 となる．この符号化は，テキストを文節木によって部分文字列に分割し，それぞれに符号語を割り当てることで得られる．また，分割された各部分文字列をブロックと呼ぶ．たとえば，この例では，符号語 011 はブロックacb を表現している．

今，テキストが記憶のない情報源からの系列であると仮定しよう．このとき，ブロックの平均長を最大にするという意味で，大きさm の最適な文節木$\mathcal{T}_m^*$ は，以下のようにして構築できる．

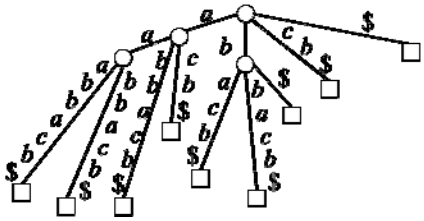

図 2. 接尾辞木の例(T=aaabbacb)

仮定より, 情報源記号$a \in \Sigma(k = |\Sigma|)$の出現確率を$Pr(a)$とすると，文節木の根から節点$\mu$へ至るパスを辿る文字列$x_\mu \in \Sigma^+$の出現確率は$Pr(x_\mu) = \prod_{\eta \in \xi} Pr(\eta)$となる．ここで，$\xi$ は根からμ までのパス上のラベル列である．根から各葉へ至る文字列の出現確率を，その葉の確率と呼ぶことにすると，最適な文節木$\mathcal{T}_m^*$ を構築する手順は以下のとおりである．

1. 大きさ1 の最適な文節木$\mathcal{T}_m^*$ は, 根とk個の子からなる深さ1 の順序付きk分木$\mathcal{T}_1$である．この$\mathcal{T}_1$を初期木と呼ぶ．
2. 各$i = 2, \ldots, m$について，以下を繰り返す．

（ア）大きさ$i-1$ の文節木$\mathcal{T}_{i-1}^*$の葉のうち，最大の確率を持つ葉$v = v_i^*$ を選択する．

（イ）初期木$\mathcal{T}_1$をv_i^* に接ぎ木し, それを大きさi の文節木$\mathcal{T}_i^*$とする．

任意の整数$m \geq 1$について，上の手順で構築された文節木$\mathcal{T}_m^*$を Tunstall 木と呼ぶ．また，Tunstall 木$\mathcal{T}_m^*$を用いた VF 符号を Tunstall 符号と呼ぶ．

文節木$\mathcal{T}_m^*$の葉の総数Nは$m(k-1)+1$なので，符号語長ℓは条件$N = m(k-1) \leq 2^\ell$ を満たす必要がある．したがって，テキストを符号語長ℓで符号化するならば，大きさ$m = \lfloor (2^\ell - 1)/(k-1) \rfloor$ の文節木$\mathcal{T}_m^*$ を構築すればよい．

Tunstall符号化されたテキストを復号化するには，符号時に用いた文節木が必要である．よって，圧縮テキストは，文節木に関する情報を持たなくてはならない．一般の木の簡潔な表現方法については，既に効率良い手法が提案されている[18][19][20]．ただし，それらはn節点の木に対して$2n + o(n)$ビットを必要とする. 順序付きk文木に限るならば，より効率良い表現方法として前順符号化(preorder coding)[21]がある．前順符号化では，n節点の木に対してちょうどnビットで木を符号化できる．ただし，別途アルファベットサイズkの情報を保持する必要がある．この符号化は，対象の木を根から深さ優先順で探索しつつ，到達した節点が内部節点なら 1 を, 葉なら 0 を出力することで符号化を行う．したがって，たとえば図 1 の文節木は，1100100000 と符号化される．大きさmのk分木に対して，出力されるビット総数は$n = mk + 1$である．前順符号化は，木がそれほど大きくないときには価値がある．Tunstall木は，情報源記号の出現確率が分かれば復元できるので，木が非常に大きくなった場合には，前順符号化よりも出現確率表を保持したほうがよい．

3.3 接尾辞木

接尾辞木は，文字列Tに対してFac(T)のすべての要素をコンパクトに表現するデータ構造である（図 2）．

接尾辞木は四つ組$ST(T) = (V, root, E, suf)$で表される．ここで, Vは節点の集合, Eは辺の集合で$E \subseteq V^2$である．ST(T)に対し，グラフ(V, E)は$root \in V$を頂点とする根付き木を構成する．すなわち，根 $root$ から各節点$s \in V$へのパスはただ一つ存在する．ある節点$s, t \in V$について$(s, t) \in E$であれば，節点sをtの親と呼び，節点t をs の子と呼ぶ．また，節点が子を持つならば内部節点と呼び，子を持たないならば葉と呼ぶ．さらに，根$root$から$s \in V$へのパス上にある各節点をsの祖先と呼ぶ．

Eの各辺はTの部分文字列$T[j:k]$でラベル付けされており，その文字列をラベル文字列と呼ぶ．実際には二つの整数の組(j, k)で表現される．節点$s \in V$へ向かう辺のラベル文字列を$label(s)$とする．任意の子はただ一つの親を持つので，$label(s)$はユニークに定義される．任意の節点$s \in V$について，根$root$からsへのパス上にあるすべての辺のラベル文字列を連結したものを$\bar{s}$とする．また，これをsが表現する文字列と呼ぶ．すなわち，$root, a_1, a_2, \ldots, s$ をs の祖先の列とすると，$\bar{s} = label(root) \cdot label(a_1) \cdot label(a_2) \cdots label(s)$である．

与えられるテキストTについて, ST(T\$)の各葉はTの各接尾辞に一対一対応している．また，すべての内部節点は二つ以上の子を持つ．ここで，\$は終端記号と呼ばれる記号で，Σには含まれないものとする．このとき，接尾辞木ST(T\$)には$|T|$個の葉が存在し, $2|T| - 1$以下の節点が含まれる. 各節点$s \in V$において任意の二つの子x, yに対し，$\bar{x}$と$\bar{y}$は，必ず異なる文字から始まる（$\bar{x}[1] \neq \bar{y}[1]$）．

以上の議論から，各節点$s \in V$はTの一つの部分文字列に一対一に対応することが証明できる．

4. STVF 符号

接尾辞木ST(T\$)は，その最も深い葉がテキストT全体を表しているので，そのままでは文節木として用いることができない．我々のアイデアは，接尾辞木ST(T\$)を適切に短く刈り込むことで，コンパクトな文節木を構築することである．

接尾辞木ST(T\$)を刈り込んでできた木を刈り込み接尾辞木と呼び，その木の葉の個数がL個となったものを$ST_L(T\$)$と書くことにする．図¥ref{fig:pst}は，刈り込み接尾辞木$ST_L(T\$)$の例を示している．元の接尾辞木ST(T\$)において深さが1である節点すべてを持つような刈り込み接尾辞木は, T中のすべての記号を含んでいることに注意する．刈り込み接尾辞木$ST_L(T\$)$を使ってTを分割，符号化する手順は Tunstall 符号と同じである．

今，符号語長ℓでテキストTを符号化するとしよう．接尾辞木中のどのような文字列に符号語を割り当てるかによって，圧縮率は変化する．符号語長は固定なので，圧縮率を良くするためには，$L \leq 2^\ell$を満たしつつ，テキストの分割数を可能な限り小さくすればよい．言い換えると，ブロックの平均長ができるだけ長くなるように接尾辞木を刈り込めば良い．ところが，刈り込んだ接尾辞木の枝の平均長が長くなるように刈り込む戦略には問題がある．深い葉に割り当てた符号語が必ずしも使用されるとは限らないからである．頻度の低い文字列を表す葉に符号語を割り当てると，使用されない可能性が高くなる．[17]で議論されているように，最適な刈り込みを求めることは困難であるため，現実的には刈り込みアルゴリズムはヒューリスティックなものを使わざるを得ない．

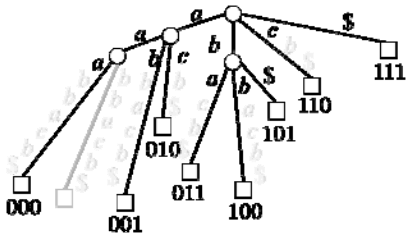

図 3. 刈り込み接尾辞木の例

最も単純な刈り込み戦略は，根のみからなる木を初期の刈り込み接尾辞木$ST_1(T\$)$とし，ST(T$)を根から幅優先探索しつつ浅い位置にある節点から順に$ST_L(T\$)$（$L \geq 1$）へ加え，葉の数Lが$L \leq 2^{\ell}$を満たさなくなったら$ST_L(T\$)$の伸長を止めることである．しかし，この戦略では，浅い位置にある頻度の低い節点を多く選択してしまう可能性が高い．

より慎重な戦略は，$ST_L(T\$)$を伸長させるときに，頻度の高い文字列を表す節点から順に選んでいくことである．言いかえると，刈り込み接尾辞木$ST_L(T\$)$の各葉が表す文字列の頻度が，なるべく一様になるように節点を選ぶことである．この戦略では，浅い位置にある頻度の低い節点をなるべく避けるように選択していくことができる．

符号語長ℓとテキストTが与えられたとき，以下の手順で刈り込み接尾辞木を構築する．

1. まず，接尾辞木ST(T$)を構築する．
2. ST(T$)の根とそのすべての子だけからなる刈り込み接尾辞木$ST_{k+1}(T\$)$を，初期の文節木候補$\mathcal{T}'_1$とする．
3. $\mathcal{T}'_1 = ST_{L_i}(T\$)$中の葉のうち，元のST(T$)上で最も頻度が高い文字列を表す節点vを選ぶ．また，L_iを$\mathcal{T}'_i$の葉の総数とし，vの子の数をC_vとする．
4. もし$L_i + C_v - 1 \leq 2^{\ell}$を満たすならば，vのすべての子を新たな葉として$\mathcal{T}'_i$ に加え，それを新たな文節木候補$\mathcal{T}'_{i+1}$とする．もし，vの子uが接尾辞木ST(T$)において葉であった場合には，vからuへ至る辺のラベルを先頭一文字だけを残して切り落とす．
5. ステップ 3 と 4 を，木$\mathcal{T}'_i$ が伸ばせなくなるまで繰り返す．

図 3 は，テキストT = aaabbacb$，符号語長$\ell = 3$の場合に，上の手順で構築される文節木の例である．この文節木によって，テキストTは$aa, ab, ba, c, b\$$ に分割される．

上の手順で構築された刈り込み接尾辞木$\mathcal{T}'$について，次の補題が成り立つ．

［**補題**］任意のテキストTに対して，上述の刈り込み接尾辞木$\mathcal{T}'$ を文節木として用いると，Tを一意に分割することができる．

証明．刈り込み接尾辞木$\mathcal{T}'$の各葉が表す文字列全体からなる集合をDとする（Dは辞書と呼ばれる）．$\mathcal{T}'$を文節木として用いたテキスト分割は，辞書Dを使って分割することに等しい．また，$\mathcal{T}'$の各葉は，接尾辞木の性質より，Tのある部分文字列と一対一対応していることに注意する．

テキストT = T[1: u]の接頭辞T[1: i − 1] は既に分割・符号化されており，これから残りの接尾辞T[i: u]を分割・符号化するとしよう．次に分割されるブロックは，T[i: u]の最長の接頭辞で，かつ，辞書Dに含まれる最も長い文字列である．そのような文字列はD中にただ一つ存在する．もし二つ以上存在したとすると，刈り込み接尾辞木$\mathcal{T}'$中の複数の葉が同じ文字列を表していることになる．これは接尾辞木の性質に矛盾する．逆に，そのような文字列が一つも存在しないとすると，T[i: u]の接頭辞で長さが1以上の物がDに登録されていないことになる．しかしながら，上で述べた刈り込み接尾辞木構築アルゴリズムのステップ 1 で，初期の文節木候補を$ST_{k+1}(T\$)$としており，これはST(T$)の深さ1の節点すべてを含む木である．よって，Dは少なくともテキスト中の任意の文字（から始まる文字列）を含むため，矛盾する．以上から，命題は成り立つ．〈証明終〉

4. VF 符号上のパターン照合

今，テキストT = T[1: u]が，ある VF 符号によって符号語の列$Z = Z[1: n] = c_1, c_2, \ldots, c_n$に圧縮されているとする．各符号語$c_i$ $(1 \leq i \leq n)$ は，テキストTのある部分文字列に対応していることに注意する．符号語c_iに対応する文字列を$w(c_i) \in D$と書くことにする（Dは辞書で，前節の証明で定義された文字列の集合）．

VF 符号上での圧縮照合問題を次のように定義する．

[問題] パターンP = P[1: m]と VF 符号による圧縮テキスト$Z = c_1, c_2, \ldots, c_n$ が与えられたとき， Zを復元して得られる元のテキスト$T[1: u] = w(c_1) \cdots w(c_n)$中に出現するパターンPの位置をすべて答えよ．

非圧縮テキストに対するよく知られた照合アルゴリズムであるKMP法を拡張し，圧縮テキスト上で動作する汎用的なアルゴリズムが[22]で示されており，VF符号はこの枠組みに当てはまる．スペースの都合より，詳細は省略するが，[22]の結果から，VF符号について，以下の定理が得られ，KMP型の照合アルゴリズムを構成することができる．

［**定理**］([22]の変形） 任意のVF符号について，それにより圧縮されたテキスト上での圧縮照合問題は， $O(m^2 + |D|)$時間・領域の前処理の後，$O(n + R)$時間で解くことができる．ここで，nは圧縮テキストの長さ，Rはパターンの出現数，mはパターンの長さ，|D|はVF符号の文節木のすべてのラベルの長さの総和である．

5. 実験結果

Tunstall符号およびSTVF符号を実装し，圧縮率の比較実験を行った．比較した手法は，Huffman符号，BPE法（[5]による照合速度向上のための技法を取り入れたもの），Tunstall符号，STVF符号の 4 つの圧縮手法である．

使用したテキストデータは，Canterburyコーパス[1]と日本語コーパスJ-TEXTS[2]およびランダムに生成したテキストの 3 種類から選択した．各々の詳細は**表 1**のとおりである．

[1] http://corpus.canterbury.ac.nz/descriptions/

[2] http://www.j-texts.com/

表 1. 使用テキストデータ

テキスト	サイズ(Byte)	|Σ|	内容
E.coli	4638690	4	E.Coli バクテリアのゲノムデータ
bible.txt	4047392	63	King James 版聖書
world192.txt	2473400	94	The CIA world fact book
dazai.utf.txt	7268943	141	太宰治全集(UTF-8)
1000000.txt	1000000	26	自動生成されたランダムテキスト

表 2. 各圧縮法による圧縮結果

圧縮法の名前の右側の括弧内の数字は符号語長ℓの値を示す．データサイズの単位はバイト．
dazai.utf.txt に対して，BPE のプログラムは正常に終了しなかったため，空欄としている．

	E.coli	bible.txt	world192.txt	dazai.utf.txt	1000000.txt
Huffman	**25.00%**	54.82%	63.03%	57.50%	**59.61%**
BPE	29.70%	46.95%	56.85%	-	70.61%
Tunstall(8)	27.39%	72.70%	85.95%	100.00%	76.39%
Tunstall(12)	26.47%	64.89%	77.61%	69.47%	68.45%
Tunstall(16)	26.24%	61.55%	70.29%	70.98%	65.25%
STVF(8)	25.09%	66.59%	80.76%	73.04%	74.25%
STVF(12)	25.10%	50.25%	62.12%	52.99%	68.90%
STVF(16)	28.90%	**42.13%**	**49.93%**	**41.37%**	78.99%
gzip	28.91%	29.43%	29.30%	33.41%	63.53%

表 3. 圧縮照合速度の比較結果

単位は秒．時間が 0 ものは，計測できないくらい短い時間であることを示している．
照合アルゴリズムの名前の後ろの括弧内の数字は，符号長を表している．

	テキスト	E.coli	bible.txt	world192.txt
PMM on Huffman	前処理	0.000	0.000	0.000
	走査	0.723	1.394	0.976
PMM on Tunstall (8)	前処理	0.000	0.003	0.000
	走査	0.015	0.031	0.025
PMM on Tunstall (16)	前処理	0.277	0.306	0.302
	走査	**0.006**	**0.015**	0.012
PMM on STVF (8)	前処理	0.003	0.003	0.006
	走査	0.012	0.018	0.015
PMM on STVF (16)	前処理	0.633	0.546	0.540
	走査	0.009	0.016	**0.009**

圧縮結果は**表 2**のとおりである（参考までにgzipの圧縮率も表に含めた）．STVF符号について，圧縮ファイルは刈り込み接尾辞木の情報も含んでいる．ただし，今回の実装では，木の構造をバランスした括弧によって符号化[20]したものに加えて，ラベル文字列を無圧縮で出力している．この木の情報をコンパクトに表現することができれば圧縮率の向上につながる．

表 2からは，アルファベットサイズが小さくないところで，STVF符号がTunstall符号化やHuffman符号化よりも圧縮率が良いことが分かる．

次に，圧縮照合の速度比較について実験を行った．比較を行ったのはHuffman符号上，Tunstall符号上，STVF符号上でのKMP型アルゴリズムである．実験環境は，Xeon® プロセッサ 3.00GHzデュアルコア，メモリ 16GB，Windows Vista 上のcygwin環境である．プログラムはすべてGNU g++ version 3.4.4 でコンパイルされている．Huffman符号に対するパターン照合アルゴリズムは[23]による実装を用いた．対象テキストは，**表 1**のうちからE.coli，bible.txt，world192.txtを選択した．各テキストについて，長さ 3〜11 のパターンを 5 つ選択し，その照合速度の平均を取った．なお，今回，[5]で提案されているBPE上のBM型アルゴリズムは，同じ環境で動作しなかったため，比較からは外した．実験結果は**表 3**のとおりである．

Huffman符号上の照合では，符号語の境目を処理するため，ビット単位でのデータの取り扱いが必要であることに対し，Tunstall符号やSTVF符号上での照合は，バイト単位で処理できるため，高速に照合できる．**表 3**から，符号長が 16 のときのほうが，Tunstall符号・STVF符号双方とも，テキストの走査速度が速い．ただし，前処理により多くの時間がかかっており，STVF符号のほうが，Tunstall符号よりも前処理にさらに多くの時間がかかっている．これは，刈り込み接尾辞木を復元するための時間コストが大きいためと思われる．テキスト走査時間だけでいえば，圧縮率の高いほうが有利であることが考察できる．

6. おわりに

本論文では，STVF 符号と名づけた新しい VF 符号について提案し，Huffman 符号や Tunstall 符号より優れた圧縮率を達成することを示した．また，Tunstall 符号や STVF 符号上での圧縮照合は Huffman 符号上の照合よりも格段に高速であることを示した．圧縮照合に適した性質を持ちつつ，Tunstall 符号よりも大幅な圧縮率改善を達成したことは，実用上有意義である．

現時点では，テキストに対する接尾辞木を一旦構築する必要があるため，大規模なテキストに適用しにくいという問題がある．大規模なテキストに対し，動的に刈り込みを行いつつ文節木を構築することで対応できるが，具体的なアルゴリズムの開発は今後の課題である．また，STVF 符号上での圧縮照合処理において，前処理時間を低減する手法の開発も重要である．

[謝辞]

本研究は，日本学術振興会科学研究費補助金（若手研究：20700001）の補助を受けています．

[文献]

[1] Crochemore, M. and Rytter, W.: “Jewels of Stringology”, World Scientific Publishing (2002).
[2] Salomon, D.: “Data Compression: The Complete Reference”, 4th edition, Springer (2006).
[3] Sayood, K.: “Lossless Compression Handbook”, Academic Press (2002).
[4] Amir, A. and Benson, G.: "Efficient two-dimensional compressed matching", Proc. of DCC’92, pp.279-288 (1992).
[5] Shibata, Y., Matsumoto T., Takeda, M., Shiohara, A. and Arikawa, S.: “A Boyer-Moore type algorithm for compressed pattern matching”, In Proc. of 11st Annual Symposium on Combinatorial Pattern Matching (CPM 2000), LNCS 1848, pp. 181-194 (2000).
[6] Rautio, J., Tanninen, J. and Tarhio, J.: “String Matching with Stopper Encoding and Code Splitting”, In Proc. of 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002), LNCS 2373, pp. 42-52 (2002).
[7] Tunstall, B. P.: “Synthesis of noiseless compression codes”, Georgia Inst. Technol., Atlanta, GA (1967).
[8] Ziv, J.: “Variable-toFixed Length Codes are Better than Fixed-to-Variable Length Codes for Markov Sources”, IEEE Transactions on Information Theory, 36(4), pp. 861-863, July (1990).
[9] Tjalkens, T. J. and Willems, F. M. J.: “Variable to Fixed-Length Codes for Markov Sources”, IEEE Trans. on Information Theory, IT-33(2), Mar. (1987).
[10] Savari, S. A. and Gallager, R. G.: “Generalized Tunstall codes for sources with memory”, IEEE Transactions on Information Theory, 43(2), pp. 658-668, Mar. (1997).
[11] Abrahams, J.: “Code and parse trees for lossless source encoding”, Compression and Complexity of Sequences 1997, pp. 145-171, Jun. (1997).
[12] Visweawariah, K., Kulkarni, S. R. and Verdú, S.: “Universal Variable-toFixed Length Source Codes”, IEEE Trans. on Information Theory, 47(4), pp. 1461-1472, May (2001).
[13] Yamamoto, H. and Yokoo, H.: “Average-Sense Optimality and Competitive Optimality for Almost Instantaneous VF Codes”, IEEE Trans. on Information Theory, 47(6), pp. 2174-2184 (2001).
[14] 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), LNCS 5280, pp. 27-38, Nov. (2008).
[15] Kida, T.: “Suffix Tree Based VF-Coding for Compressed Pattern Matching”, Proc. of Data Compression Conference 2009(DCC2009), p. 449, Mar. (2009).
[16] 喜田拓也: “頻度刈り込み接尾辞木による VF 符号化”, DEIM Forum 2009 E1-4, Mar. (2009).
[17] Klein, S. T. and Shapira, D.: “Improved Variable-to-Fixed Length Codes”, Proc. of 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008), LNCS 5280, pp. 39-50, Nov. (2008).
[18] Jansson, J., Sadakane, K. and Sung, W.: “Ultra-succinct representation of ordered trees”, SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 575-584 (2007).
[19] Benoit, D., Demaine, E. D., Munro, J. I., Raman, R., Raman, V. and Rao, S. S.: “Representing Trees of Higher Degree”, Algorithmica, 43(4), Springer-Verlag, pp. 275-292 (2005).
[20] Munro, J. I.: “Space efficient suffix trees”, J. Algorithms, 39(2), Academic Press, pp. 205-222 (2001).
[21] Kobayashi, K. and Han, T. S.: “On the Pre-order Coding for Complete k-ary Coding Trees”, In Proc. of Inter. Symp. on Information Theory and Its Applications, pp. 302-303 (1996).
[22] Kida, T., Shibata, Y., Takeda, M., Shinohara, A. and Arikawa, S.: “A Unifying Framework for Compressed Pattern Matching", Proc. of 6th International Symp. on String Processing and Information Retrieval, IEEE Computer Society, pp. 89-96 (1999).
[23] Takeda, M., Shibata, Y., Matsumoto, T., Kida, T., Shinohara, A., Fukamachi, S., Shinohara, T. and Arikawa, S: "Speeding Up String Pattern Matching by Text Compression: The Dawn of a New Era", Transactions of Information Processing Society of Japan, IPSJ, 42(3), pp. 370-384 (2001).

喜田 拓也 Takuya KIDA

北海道大学大学院情報科学研究科准教授．2001 九州大学大学院システム情報科学研究科博士後期課程修了，博士（情報科学）. 2001 九州大学附属図書館, 研究開発室専任講師. 2004 年より現職．テキストアルゴリズムおよび情報検索技術に関する研究・開発に従事．情報処理学会正会員．電子情報通信学会正会員．日本データベース学会正会員．