頻度刈り込み接尾辞木によるVF符号化

喜田　拓也†

† 北海道大学大学院情報科学研究科　〒 060–0814 札幌市北区北 14 条西 9 丁目
E-mail: †kida@ist.hokudai.ac.jp

あらまし　本稿では，接尾辞木を用いた新たな VF 符号化手法を提案する．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，Tunstall 符号や Huffman 符号よりも圧縮性能が良いことがわかった．
キーワード　VF 符号，接尾辞木，Tunstall 符号，圧縮パターンマッチング

VF Coding Using Frequency-base-pruned Suffix Tree

Takuya KIDA†

† Graduate School of Information Science and Technology, Hokkaido University　Kita 14 Nishi 9, Sapporo, 060–0814 Japan
E-mail: †kida@ist.hokudai.ac.jp

Abstract　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.
Key words　VF code, suffix tree, Tunstall code, compressed pattern matching

1. はじめに

テキスト圧縮 [3] は，テキスト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 Run-length 法，LZ 系圧縮法，BW 変換に基づく手法など，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 [12], [16].

一方で，圧縮技術と検索照合技術とを組み合わせる研究が 90 年代初頭に提案され [2]，**圧縮照合問題** (compressed matching problem) と呼ばれている．これは，圧縮されたテキストを元に戻さずに文字列照合を行うという問題であり，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重要な課題となっている．形式的には，テキスト文字列 $T = t_1 \dots t_u$ が圧縮された形 $Z = z_1 \dots z_n$ で存在し，パターン文字列 P 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である．単純な方法としては，まず最初に 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，復元にも時間がかかる．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ことである．ここで，R はパターンの出現回数である．しかしながら，テキストを圧縮することと検索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，効率良く圧縮照合できるいくつかの方法が出現した [11], [17]．それらは圧縮後に照合する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ことにある．そのような圧縮法には，共通して次のような性質を備えている．

- 固定長符号である．特にバイト単位の符号化が望ましい．
- 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，圧縮率の高い符号化手法を開発することが求められる．

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．一方，**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する．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** [19] 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．

本稿では，Tunstall 符号の実際の圧縮率を報告するとともに，VF 符号の枠組みで良い圧縮率を実現する新たな符号化手法を提案する．提案 VF 符号 (**ST-VF 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) 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，ST-VF 符号および Tunstall 符号を実際に実装し，その圧縮率に関する評価実験を行った結果を示す．

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 [22]．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 [18]．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 [13]．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 [1]．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 [20]．

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 [21]．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，情報源記号上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 [6], [10], [11], [17]．特にバイトペア符号化法 (BPE 法) や Stopper Encoding 法 (SE 法) 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 [8]．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．

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，$\Sigma^+ = \Sigma^* - \{\varepsilon\}$ 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 $m \geqq 1$ 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 2 元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割して得られる各部分文字列を**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する．情報源記号 $a \in \Sigma (k = |\Sigma|)$ の出現確率を $Pr(a)$ とする

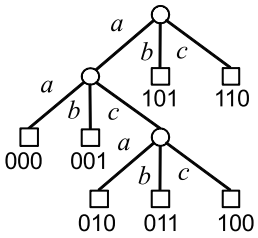

図 1　文節木の例

と，このとき，文節木の根から節点 μ 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，ξ は根から μ までのパス上のラベル列である．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，次のようにして構築できる．

（1） 大きさ 1 の最適な文節木 $\mathcal{T}_1^*$ 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．

（a） 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．

（b） 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 $m \geqq 1$ 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 $N = m(k-1) \leqq 2^\ell$ 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 [4], [5], [9]．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) [7] 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．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 (suffix tree) 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．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 $E \subseteqq V^2$ 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 $\overline{s}$ 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，$\overline{s} = label(root) \cdot label(a_1) \cdot label(a_2) \cdots label(s)$ 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，$\overline{x}$ と $\overline{y}$ は，必ず異なる文字から始まる（$\overline{x}[1] \neq \overline{y}[1]$）．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. ST-VF 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 2 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．今，符号語長 ℓ でテキスト T を符号化するとしよう．Tunstall 符号と同様に，条件式 $L \leqq 2^\ell$ は満たされなくてはならない．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．

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 $ST_L(T\$)$（$L \geqq 1$）へ加え，葉の数 L が $L \leqq 2^\ell$ 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．

より慎重な戦略は，$ST_L(T\$)$ を伸長させるときに，最も頻度の高い文字列を表す節点を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．したがって，

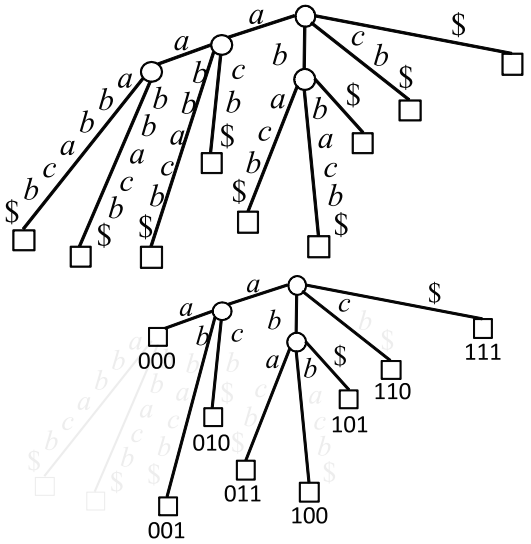

図 2　接尾辞木と刈り込み接尾辞木の例

この戦略による接尾辞木を刈り込むアルゴリズムは次のようになる．

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） $\mathcal{T}_i' = ST_{L_i}(T\$)$ 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 $L_i + C_v - 1 \leqq 2^\ell$ 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 2 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** 1］ 刈り込み接尾辞木 $\mathcal{T}'$ を文節木として用いると，任意のテキスト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． □

5. 実験結果

Tunstall 符号および ST-VF 符号を実装し，圧縮率の比較実験を行った．ここでは，3 つの圧縮手法（Huffman 符号，Tunstall 符号，ST-VF 符号）について比較を行う．

使用したテキストデータは，Canterbury コーパス(注1)と日本語コーパス J-TEXTS(注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．

圧縮結果は表 2 のとおりである．ST-VF 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 [9] 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（すなわち自然言語によるテキスト）で，ST-VF 符号が他の符号化より優れていることが分かる．

6. おわりに

本稿では，ST-VF 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．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．本稿では省略したが，ST-VF 符号上での圧縮パターン照合アルゴリズムは既に得ており，今後圧縮照合速度の比較実験を行う準備を行っている．

一方，圧縮率の向上を第一の目的とした場合，ST-VF 符号で得られる系列に Huffman 符号などの FV 符号を施すことが考えられる．これは，すなわち，非等長情報源の非等長符号化（VV 符号）に他ならない．その圧縮率の実証実験も，今後の課題の一つである．

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．

(注1)：http://corpus.canterbury.ac.nz/descriptions/
(注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)	$\|\Sigma\|$	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 Byte.

圧縮法	E.coli	bible.txt	world192.txt	dazai.utf.txt	1000000.txt
データサイズ	4638690	4047392	2473400	7268943	1000000
Huffman	**25.00%**	54.82%	63.03%	57.50%	**59.61%**
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%
ST-VF(8)	25.09%	66.59%	80.76%	73.04%	74.25%
ST-VF(12)	25.10%	50.25%	62.12%	52.99%	68.90%
ST-VF(16)	28.90%	**42.13%**	**49.93%**	**41.37%**	78.99%

文　献

[1] Julia Abrahams. Code and parse trees for lossless source encoding. In *Compression and Complexity of Sequences 1997*, pages 145–171, Jun. 1997.

[2] A. Amir and G. Benson. Efficient two-dimensional compressed matching. In *Proc. DCC'92*, pages 279–288, 1992.

[3] T. Bell, J. Cleary, and I. Witten. *Text Compression*. Prentice Hall, 1990.

[4] David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.

[5] Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 575–584, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[6] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In *Proc. DCC'98*, pages 103–112, 1998.

[7] K. Kobayashi and T. S. Han. On the pre-order coding for complete k-ary coding trees. In *In Proc. of Inter. Symp. on Information Theory and Its Applications*, pages 302–303, 1996.

[8] Shirou Maruyama, Yohei Tanaka, Hiroshi Sakamoto, and Masayuki Takeda. Context-sensitive grammar transform: Compression and pattern matching. In *Proc. 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008)*, Nov. 2008. (to appear).

[9] J. Ian Munro. Space efficient suffix trees. *J. Algorithms*, 39(2):205–222, 2001.

[10] G. Navarro and M. Raffinot. A general practical approach to pattern matching over Ziv-Lempel compressed text. In *Proc. CPM'99*, LNCS 1645, pages 14–36, 1999.

[11] Jussi Rautio, Jani Tanninen, and Jorma Tarhio. String matching with stopper encoding and code splitting. In *In Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002)*, LNCS 2373, pages 42–52, 2002.

[12] David Salomon. *Data Compression: The Complete Reference*. Springer, 3rd edition, 2004.

[13] S. A. Savari and R. G. Gallager. Generalized tunstall codes for sources with memory. *IEEE Transactions on Information Theory*, 43(2):658–668, Mar. 1997.

[14] Serap A. Savari. Variable-to-fixed length codes for predictable sources. In *In Proc. of DCC98*, pages 481–490, 1998.

[15] Serap A. Savari. Variable-to-fixed length codes and plurally parsable dictionaries. In *In Proc. of DCC99*, pages 453–462, 1999.

[16] Khalid Sayood, editor. *Lossless Compression Handbook*. Academic Press, 2002.

[17] Y. Shibata, T. Matsumoto, M. Takeda, A. Shiohara, and S. Arikawa. A Boyer-Moore type algorithm for compressed pattern matching. In *In Proc. 11st Annual Symposium on Combinatorial Pattern Matching (CPM 2000)*, LNCS 1848, pages 181–194, 2000.

[18] Tjalling J. Tjalkens and Frans M. J. Willems. Variable to fixed-length codes for markov sources. *IEEE Trans. on Information Theory*, IT-33(2), Mar. 1987.

[19] B. P. Tunstall. *Synthesis of noiseless compression codes*. PhD thesis, Georgia Inst. Technol., Atlanta, GA, 1967.

[20] Karthik Visweawariah, Sanjeev R. Kulkarni, and Sergio Verdú. Universal variable-tofixed length source codes. *IEEE Trans. on Information Theory*, 47(4):1461–1472, May 2001.

[21] Hirosuke Yamamoto and Hidetoshi Yokoo. Average-sense optimality and competitive optimality for almost instantaneous vf codes. *IEEE Trans. on Information Theory*, 47(6):2174–2184, Sep. 2001.

[22] Jacob Ziv. Variable-tofixed length codes are better than fixed-to-variable length codes for markov sources. *IEEE Transactions on Information Theory*, 36(4):861–863, July 1990.