Tunstall-Huffman 符号化の効率について

松井　雄大†　　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: †{yudai,kida}@ist.hokudai.ac.jp

あらまし　本稿では，Tunstall 符号と Huffman 符号を組み合わせた Tunstall-Huffman 符号の圧縮率について考察する．この符号は，Tunstall 符号化によるテキスト分割と Huffman 符号による非等長符号化を組み合わせたものである．今回，Tunstall-Huffman 符号を実際に実装し，性能比較実験を行った．実験の結果，自然言語文書などに対して Tunstall-Huffman 符号は約 43 ∼ 50% の圧縮率を達成しており，単純な Tunstall 符号や Huffman 符号よりも 10 ∼ 20% 以上の圧縮率向上が得られることがわかった．

キーワード　VF 符号，Tunstall 符号，Huffman 符号，ST-VF 符号

Study on Efficiency of Tunstall-Huffman Code

Yudai MATSUI† and Takuya KIDA†

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

Abstract　In this paper, we study the compression ratio of Tunstall-Huffman code, which is a variable-to-variable length method. VF codes as typified by Tunstall code are paid less attention to so far as compared to FV code such as Huffman code due to its mild compression ratio. We implemented and experimented Tunstall-Huffman code. The experimental results show that it achieves the compression ratio of about 43 ∼ 50% for a natural language text, which is better than simple Tunstall and Huffman codes.

Key words　VF code, Tunstall code, Huffman code, ST-VF code

1. はじめに

テキスト圧縮 [2] 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 [8], [11].

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** [13] 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，事実，本稿で示すように，その漸近する速度は非常に緩やかであり，実際のテキストに対しては Huffman 符号ほどの圧縮率は得られない．

Tunstall 符号によって分割された非等長の情報源系列に対して，Huffman 符号による非等長の符号を割り当てることは，圧縮率を向上させるための自然な考えである．しかしながら，このような **VV 符号**（Variable-length-to-Variable-length code）に関する研究は，Savari と Szpankowski ら [10] 以前にはほとんどなされていなかった．彼らによって Tunstall-Huffman 符号の冗長度の理論的上限は与えられたものの，実際のデータに対してどの程度の圧縮率を有するのかについては報告されていない．

本稿では，Tunstall 符号の実際の圧縮率を示すとともに，Tunstall-Huffman 符号を実際に実装し，その圧縮率に関する評価実験の結果を示す．

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[16]．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[12]．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[9]．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[14]．

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[15]．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．

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 $x_1 x_2$ 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．

3.2 Tunstall 符号

$\Sigma = \{a_1, \ldots, a_k\}$ を大きさ $k \geqq 1$ 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．

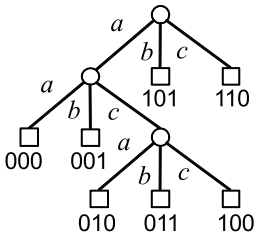

図 1　文節木の例

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となる．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，既に効果的な手法が提案されている [3], [4], [7]．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) [6] 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はある．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 Fig. 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．

表 1　使用テキスト (The Artificial Corpus)

テキスト	サイズ (byte)	$\lvert\Sigma\rvert$	内容
aaa.txt	100000	1	a のみを並べたもの
alphabet.txt	100000	26	$a \sim z$ までを繰り返したもの
random.txt	100000	26	$a \sim z$ をランダムに並べたもの

3.3　Tunstall-Huffman 符号

先に述べたとおり，Tunstall 符号の出力は Tunstall 木の葉の番号の列であり，一つ一つが元テキストの可変長の部分文字列に対応している．Tunstall 木 $\mathcal{T}_m^*$ を大きくすれば（m を大きくすれば）するほどブロックの平均長は長くなるが，コードを割り当てるべき葉の個数が増大し，一つの符号語長が長くなる．一方で，符号化後の系列に出現する番号の種類は，葉の総数と比べてその割合が少なくなっていく．すなわち，系列に含まれる符号語の偏りが増大する．したがって，それらに等長の符号語を割り当てるよりは，Huffman 符号などを用いて可変長の符号語を割り当てることで圧縮率の向上が見込める．

Savari と Szpankowski らは，VF 符号と FV 符号を組み合わせてできる VV 符号についてその理論的性能を解析している [10]．特に，Tunstall 符号と，Huffman 符号および Shannon-Fano 符号を組み合わせた方式について解析し，次のような結果を導いている．

［**定理** 1］（[10] の定理 1）

$$\limsup_{M\to\infty} \log_2 M \cdot R_{\text{T-SF}}(M) \ < \ \mathcal{H}$$

$$\limsup_{M\to\infty} \log_2 M \cdot R_{\text{T-H}}(M) \ < \ \mathcal{H}\log_2(2e^{-1}\log_2 e)$$

ここで，$\mathcal{H}$ は無記憶 2 元情報源のエントロピーで，$R_{\text{T-SF}}$ および $R_{\text{T-H}}$ は，それぞれ Tunstall-Shannon-Fano 符号と Tunstall-Huffman 符号の冗長度である．

4.　実 験 結 果

Tunstall 符号および Tunstall-Huffman 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，Tunstall-Huffman 符号）について比較を行う．

使用したテキストデータは，“The Canterbury Corpus”(注1) のサイトより，The Artificial Corpus，The Canterbury Corpus，The Large Corpus の 3 種類を用いた．各々の詳細は表 1，2，3 のとおりである．

表 4 は，The Artificial Corpus に対する圧縮結果である．アルファベットサイズが小さく，人工的に生成されたデータでは，基本的に Huffman 符号が最も良い性能を示す．ただし，alphabet.txt のように繰り返しが多いデータに対しては，Tunstall-Huffman 符号の圧縮率が非常に良くなっている．

表 5 は，The Canterbury Corpus に対する圧縮結果である．たいていの場合で，Tunstall 符号は，Huffman 符号と比べて約 10% 以上圧縮率が劣っていることが分かる．特に符号語長 $\ell = 16$ では，辞書である Tunstall 木が非常に大きくなるため，

(注1)：http://corpus.canterbury.ac.nz/descriptions/

表 2　使用テキスト (The Canterbury Corpus)

テキスト	サイズ (byte)	$\lvert\Sigma\rvert$	内容
alice29.txt	152089	74	英文小説
asyoulik.txt	125179	68	シェークスピア原作の脚本
cp.html	24603	86	HTML ソースコード
fields.c	11150	90	C 言語のソースコード
grammar.lsp	3721	76	Lisp 言語のソースコード
kennedy.xls	1029744	256	Excel データファイル
lcet10.txt	426754	84	技術文書
plrabn12.txt	481861	81	詩集
ptt5	513216	159	CCITT のテストセット
sum	38240	255	SPARC の実行ファイル
xargs.1	4227	74	GNU のマニュアルファイル

表 3　使用テキスト (The Large Corpus)

テキスト	サイズ (byte)	$\lvert\Sigma\rvert$	内容
bible.txt	4047392	63	King James 版聖書
world192.txt	2473400	94	The CIA world fact book

表 4　The Artificial Corpus に対する圧縮率
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．T-Huffman は，Tunstall-Huffman 符号を指す．

圧縮法	aaa.txt	alphabet.txt	random.txt
データサイズ	1000000	1000000	1000000
Huffman	**0.01**%	59.69%	**75.19**%
Tunstall(8)	0.44%	84.65%	96.81%
Tunstall(12)	0.56%	69.75%	76.63%
Tunstall(16)	5.12%	72.30%	97.21%
T-Huffman(8)	0.05%	48.18%	75.66%
T-Huffman(12)	0.53%	**21.71**%	89.25%
T-Huffman(16)	8.21%	27.18%	129.53%

fields.c や grammar.lsp など，数 KByte 程度のファイルでは元データよりも圧縮後のデータの方が大きくなっている．

一方，Tunstall-Huffman 符号については，Tunstall 符号単体の場合に比べ，ほとんどの場合で圧縮率が 10% から 20% 程度の向上している．ただし，入力パラメータ ℓ が 16 では，Tunstall 木が符号語系列と比較して大きくなりすぎてしまい，Tunstall 符号と同様に圧縮率の低下がみられる．

表 6 は，比較的ファイルサイズの大きい，The Large Corpus に対する圧縮結果である．符号語長 ℓ による圧縮率の変化をより細かく見るため，$\ell = 7 \sim 16$ までの圧縮率を示してある．bible.txt のように，数 MByte 以上あるような比較的多きなファイルに対しては，保存すべき Tunstall 木の大きさが相対的に小さくなるので，ℓ の値を大きくとることでより良い圧縮率が得られていることがわかる．

5.　お わ り に

本稿では，Tunstall-Huffman 符号の圧縮性能について実験を行い，考察した．実験の結果，Tunstall-Huffman 符号は，Tunstall 符号や Huffman 符号単体よりもほとんどの場合において良い圧縮率が得られることがわかった．今後の課題としては，他の VF 符号と FV 符号の組み合わせについても調査が挙げられる．また，我々は ST-VF 符号という効率の良い新しい

表 5　The Cantarbury Corpus に対する圧縮率
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．T-Huffman は，Tunstall-Huffman 符号を指す．
kennedy.xls のアルファベットサイズは 256 であるので，Tunstall(8) では圧縮できず，プログラムの仕様上，何も出力せずに停止している．

圧縮法	alice29.txt	asyoulik.txt	cp.html	fields.c	grammar.lsp	kennedy.xls	lcet10.txt	plrabn12.txt	ptt5	sum	xargs.1
データサイズ	152089	125179	24603	11150	3721	1029744	426754	481861	513216	38240	4227
Huffman	57.81%	60.73%	66.94%	65.54%	**64.77**%	45.00%	58.78%	57.25%	20.87%	**69.26**%	**67.07**%
Tunstall(8)	77.45%	79.24%	94.23%	85.04%	78.50%	-	79.44%	76.82%	100.02%	100.12%	84.01%
Tunstall(12)	68.49%	70.63%	78.95%	80.90%	82.83%	61.79%	70.63%	68.57%	23.38%	88.97%	84.65%
Tunstall(16)	70.01%	74.01%	105.40%	144.21%	287.18%	62.02%	74.01%	67.28%	21.43%	102.36%	263.24%
T-Huffman(8)	56.06%	48.28%	**66.71**%	**65.53**%	67.08%	-	57.30%	54.74%	20.89%	69.36%	71.52%
T-Huffman(12)	**52.11**%	**44.94**%	70.86%	70.55%	91.05%	34.31%	**52.18**%	**49.72**%	**12.74**%	69.93%	99.81%
T-Huffman(16)	60.69%	54.67%	111.40%	145.22%	300.35%	**30.04**%	53.29%	51.33%	15.44%	99.67%	289.40%

表 6　The Large Corpus に対する圧縮率
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．T-Huffman は，Tunstall-Huffman 符号を指す．

圧縮法	bible.txt	world192.txt
データサイズ	4047392	2473400
Huffman	54.82%	63.03%
Tunstall(7)	70.94%	87.50%
Tunstall(8)	72.70%	85.95%
Tunstall(9)	68.28%	83.01%
Tunstall(10)	67.90%	80.51%
Tunstall(11)	65.84%	77.48%
Tunstall(12)	64.89%	77.61%
Tunstall(13)	64.10%	75.06%
Tunstall(14)	63.78%	71.93%
Tunstall(15)	62.28%	71.06%
Tunstall(16)	61.55%	70.29%
T-Huffman(7)	52.10%	63.03%
T-Huffman(8)	51.83%	61.63%
T-Huffman(9)	49.81%	60.11%
T-Huffman(10)	49.13%	58.63%
T-Huffman(11)	47.38%	56.01%
T-Huffman(12)	46.41%	54.75%
T-Huffman(13)	45.35%	53.56%
T-Huffman(14)	44.08%	52.27%
T-Huffman(15)	**43.17**%	51.63%
T-Huffman(16)	51.33%	**50.84**%

VF 符号を得ており [5]，これと FV 符号を組み合わせる準備を行っている．

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

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

[3] 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.

[4] 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.

[5] Takuya Kida. Suffix tree based VF-coding for compressed pattern matching. In *Proc. of Data Compression Conference 2009*, March 2009. (to be appeared).

[6] 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.

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

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

[9] 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.

[10] Serap A. Savari and Wojciech Szpankowski. On the analysis of variable-to-variable length codes. In *In Proc. of ISIT2002*, page 176, June 2002.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.