A-008

効率よい VF 符号化のための分節木を訓練する新手法
A new approach for training parse trees for efficient VF coding

吉田 諭史* 喜田 拓也 *
Satoshi Yoshida Takuya Kida

1 背景

テキスト圧縮において最も重視される要素は，多くの場面において，圧縮率である．そのため，符号語のビット長を巧妙に切り詰められる可変長符号化手法の研究が現在の主流である．しかし，近年，テキスト圧縮を利用してパターン照合処理を高速化するという視点から，固定長符号である VF 符号 (variable-length-to-fixed-length code) が見直されている [1, 2]．VF 符号は符号語長が固定長であるという制約から，高い圧縮率を達成することが難しい．このような背景から，高い圧縮率を達成できる VF 符号の開発が望まれている．

古典的な VF 符号である Tunstall 符号 [4] は，理論的には Huffman 符号と同等の圧縮性能であることが示されているが，符号語長に対する平均符号長のエントロピーへの漸近が非常に緩やかであり，現実的には Huffman 符号ほどの圧縮率は得られないことが判明している [6]．これまで，比較的少数ではあるが，VF 符号を改善する手法がいくつか提案されている [1–3,5]．

VF 符号の圧縮率は分節木の質と大きさによって決まる．前者については，与えられたテキストに対して，ある一定数の符号語を割り当てる分節木のうち，最適な木を得るという問題は NP 困難であることが知られている [2]．一方，分節木の大きさは符号語長に依存する．符号語長を長くすれば分節木は指数的に大きくなり，より長い文字列を一つの符号語で表現できるようになるため，テキストの分割数は少なくなる．しかし，一般に VF 符号では復号時にも分節木の情報が必要であるため，逆に分節木を保存するコストが増大する．また，巨大な分節木を構築するためには多くの時間とメモリ容量を必要とするため，実用的な観点からは分節木を際限なく大きくすることはできない．

本稿では，MDL 原理に基づいて分節木に含める文字列を貪欲に決定していく訓練アルゴリズムを提案する．すなわち，提案アルゴリズムは，分節木を再構築していく際に，符号化されたブロック系列の長さの総和と分節木を符号化したものとの和が短くなるかどうかを基準に，登録する文字列の有用性を判断する．また，同じ基準でもって自動的に符号語長も調節するため，あらかじめ設定するパラメータが不要になる．提案アルゴリズムは，入力テキスト長 N とすると，$O(N)$ 領域を使って $O(rN^2)$ 時間で動作する．ここで，r は訓練中のテキスト走査の回数であり，入力テキストを初期木で分割したときの分割数を M とすると $O(M)$ で抑えられる数である．

* 北海道大学 大学院情報科学研究科 コンピュータサイエンス専攻知識ソフトウェア科学講座 情報知識ネットワーク分野

2 準備

有限アルファベットΣ の要素を**文字**とよぶ．Σ^* は Σ 上の文字列すべてからなる集合である．集合 S の大きさを $|S|$ と書き，また，文字列 $x \in \Sigma^*$ の長さを $|x|$ と書く．特に，長さが 0 の文字列を**空語**とよび，ε で表す．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．また，$w, x, y, z \in \Sigma^*$ について，$w = xyz$ が成り立つとき，x, y, z をそれぞれ w の**接頭辞**，**部分文字列**，**接尾辞**とよぶ．文字列 w の i 番目の文字を $w[i]$ と表し，i 番目から j 番目までの文字を連続して並べた部分文字列を $w[i:j]$ と表す．便宜的に，$i > j$ の場合は $w[i:j] = \varepsilon$ と定義する．木構造のうち，子のあるノードを**内部ノード**，子のないノードを**葉ノード**（または**葉**）とよぶ．親を持たないノード（すなわち木の頂点）を**根**あるいは**ルートノード**とよぶ．

3 VF 符号

以下に，VF 符号の符号化方法について概説する．入力テキスト $T \in \Sigma^+$ に対して，これを長さ $\ell \geq 1$ ビットの符号語で VF 符号化する場合を考える．いま，L 個の葉をもつ分節木 $\mathcal{T}$ が与えられているとする．$\mathcal{T}$ の各葉は，ℓ ビットの整数で番号付けされている．ただし，$L \leq 2^\ell$ である．このとき，$\mathcal{T}$ によるテキストの符号化は，以下の手順で行われる：(1) $\mathcal{T}$ 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，分節木 $\mathcal{T}$ 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．ここで，分節木によって符号語に対応付けされた（分割された）各部分文字列を**ブロック**とよぶ．(3) ステップ 2 をテキストの終端まで繰り返す．

分節木の符号化は，木構造と辺上のラベルに分割して考える．ここでは単純に括弧列による符号化を行うとする．この場合，分節木のノード数 N に対して，$2N$ ビットで符号化できる．また，各ノードに対して，符号語をもつかどうかの情報をもたせる必要がある．この情報は 1 ビットで表すことができるので，N ビット追加で必要となる．辺上のラベルについては，分節木を前順走査した順番に並べて保存する．ラベル同士の間にはアルファベット上にない区切り文字を挟む必要があるので，一文字あたり $\lceil \log(|\Sigma| + 1) \rceil$ ビットで表現されることになる．ラベルの長さの総和を W とすると，ラベル列の保存には $(W + N - 1)\lceil \log(|\Sigma| + 1) \rceil$ ビットが必要となる[*1]．一方，テ

[*1] もちろん，ラベル系列にさらなる符号化を施すことで保存コストを削減

Algorithm TrainingParseTree
Input: テキスト $S := S[1] \dots S[N]$
Output: 分節木 $\mathcal{T}$
1: $\mathcal{T} \leftarrow$ ルートノードと Σ の要素でラベル付けされた $|\Sigma|$ 個の子からなる初期木 $\mathcal{T}_0$
2: $\mathcal{T}' \leftarrow \mathcal{T}$
3: $A \leftarrow f_{\mathrm{MDL}}(S, \mathcal{T})$
4: **do**
5: $i \leftarrow 0$
6: *trained* $\leftarrow$ **false**
7: **while** $i < N$ **do**
8: $p_1 \leftarrow S[i : N]$ の接頭辞でなおかつ $\mathcal{T}$ に登録された最長の文字列
9: $p_2 \leftarrow S[i + |p_1| : N]$ の接頭辞でなおかつ $\mathcal{T}$ に登録された最長の文字列
10: $\mathcal{T}' \leftarrow \mathcal{T}$
11: $\mathcal{T}'$ に文字列 p_1p_2 を登録する
12: $A' \leftarrow f_{\mathrm{MDL}}(S, \mathcal{T}')$
13: **if** $A' < A$ **then**
14: $\mathcal{T} \leftarrow \mathcal{T}'$
15: $A \leftarrow A'$
16: *trained* $\leftarrow$ **true**
17: **else**
18: $i \leftarrow i + |p_1|$
19: **end if**
20: **end while**
21: **while** *trained* $=$ **true**
22: **return** $\mathcal{T}$

図 1　MDL 原理に基づいて分節木に含める文字列を貪欲に決定する訓練アルゴリズム

キストの符号化系列は，分割数を P とすると ℓP ビットで表現できることは明らかである．以上の議論から，最終的に出力される圧縮テキストのサイズは，

$$\ell P + 3m + (W + N - 1)\lceil \log(|\Sigma| + 1) \rceil \quad (\text{ビット}) \qquad (1)$$

となる．すなわち，入力テキストに対して，この式 (1) を可能な限り小さくするような分節木を構築することが目標となる．

4　提案手法

提案アルゴリズムを図 1 に示す．このアルゴリズムの基本的なアイデアは，入力テキストの先頭から順次，隣接するブロックを連結し，式 (1) が小さくなるならば，連結を適用し，そうでなければその部分を連結せずに，次の 2 つのブロックの連結を試みることである．

ここで，提案アルゴリズムの詳細を述べる．このアルゴリズムの入力は，テキスト S であり，出力は，分節木 $\mathcal{T}$ である．まず，初期木として，ルートノードとその $|\Sigma|$ 個の子からなる木を構築し，これを $\mathcal{T}$ とする．次に，$f_{\mathrm{MDL}}(S, \mathcal{T})$ を，$\mathcal{T}$ を用いて S 符号化したときの式 (1) の値を返す関数とする．この関数の計算には，符号語長や分割数も必要であることに注意する．まず，初期木 $\mathcal{T}_0$ に対する $f_{\mathrm{MDL}}(S, \mathcal{T}_0)$ の値を A に代入する．次に，分節木が変化しなくなるまで以下を繰り返す．木 $\mathcal{T}'$ を $\mathcal{T}$ のコピーとし，ブロック p_1p_2 を $\mathcal{T}'$ に追加する．そして，$\mathcal{T}'$ に対する $f_{\mathrm{MDL}}(S, \mathcal{T}')$ の値を A' に代入する．もし，$A' < A$ である，すなわち，ブロック p_1p_2 を追加した場合に式 (1) の値が小さくなるならば，$\mathcal{T} \leftarrow \mathcal{T}'$ として，ブロック p_1p_2 の追加を受け入れる．そして，テキストの位置 i で始まるブロック（このときには p_1p_2 となっていることに注意）とその次のブロックとを連結することを試みる．そうでない場合，すなわち，ブロック p_1p_2 を追加した場合に式 (1) の値が小さくならないのであれば，ブロック p_1p_2 の追加はなかったことにする．そして，テキストの位置 i を p_1 だけ進めて，ブロック p_2 とその次のブロックとを連結することを試みる．

次に，このアルゴリズムの計算量について述べる．関数 f_{MDL} の計算は，入力テキストを 1 回走査する必要があるため，$O(N)$ 時間かかる．そのため，1 行目から 3 行目までは $O(N)$ 時間かかる．7 行目から 20 行目までの **while** ループは最大で N 回反復される．ここからは **while** ループ内について考察する．8 行目と 9 行目はそれぞれ $O(|p_1|)$ と $O(|p_2|)$ 時間で実行できる．また，分節木の大きさは $O(N)$ なので，10 行目は $O(N)$ 時間かかる．11 行目と 12 行目はそれぞれ $O(p_1p_2)$ 時間と $O(N)$ 時間かかる．2 つのブロック p_1 と p_2 について，$|p_1|, |p_2| < N$ が成り立つことを考えると，**while** ループは全体で $O(N^2)$ 時間かかることがわかる．したがって，4 行目から 21 行目までの **do-while** ループの反復回数（これは訓練のためにテキストを走査した回数である）を r とすると，このアルゴリズムは全体で $O(rN^2)$ 時間かかることがわかる．また，各ブロックの大きさや，分節木の大きさを $O(N)$ で抑えることができるので，このアルゴリズムの領域計算量は，$O(N)$ である．

5　おわりに

本稿では，MDL 原理に基づいて分節木に含める文字列を貪欲に決定していく訓練アルゴリズムを提案した．今後の課題としては，実際の圧縮速度が実用的なレベルを達成できるようにアルゴリズムを改善し，効率良い実装を実現することが挙げられる．

参考文献

[1] T Kida. Suffix tree based VF-coding for compressed pattern matching. In *Proceedings of the Data Compression Conference*, page 449, Mar. 2009.

[2] Shmuel T. Klein and Dana Shapira. Improved variable-to-fixed length codes. In *Proceedings of the 15th International Symposium on String Processing and Information Retrieval*, volume 5280 of *Lecture Notes in Computer Science*, pages 39–50, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] T. J. Tjalkens and F. M. J. Willems. Variable to fixed-length codes for markov sources. *IEEE Transactions on Information Theory*, IT-33(2):246–257, March 1987.

[4] B. P. Tunstall. *Synthesis of noiseless compression codes*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, 1967.

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

[6] 喜田 拓也. 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. *DBSJ Journal*, 8(1):125–130, 2009.

することは可能である．ここでは，解析の簡便のため，単純な方法で保存することにしている．