情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-DBS-148 No.21
Vol.2009-FI-95 No.21
2009/7/28

仮想的な多重分節木による効率良いAIVF符号

吉田 諭史[†1]　喜田 拓也[†1]

Yamamoto と Yokoo ら [2001] によって提案された多重分節木による AIVF 符号は，短い符号長で十分な圧縮率が得られる VF 符号（可変ブロック固定長符号化）の一つである．しかしながら，情報源アルファベットのサイズを k とすると，$k-1$ 個の分節木を構築するため，ただ一つの分節木を用いる一般的な VF 符号化と比べて，符号・復号化時に多くの時間と領域が必要となる．本稿では，多重分節木を一つの分節木に統合し，その木の上で仮想的に多重分節木を模倣するアルゴリズムを示す．これにより，構築すべき分節木の総ノード数を $\Omega(k^2)$ 個，削減することができる．

Efficient AIVF Codes with Virtual Multiple Parse Tree

Satoshi Yoshida[†1] and Takuya Kida[†1]

The AIVF code with multiple parse trees presented by Yamamoto and Yokoo[2001] is one of the variable-length-to-fixed-length codes (VF codes), and it can achieve a good compression ratio even in a short codeword length. However it needs much time and space for both encoding and decoding than an ordinary VF code using just one parse tree, since it constructs $k-1$ parse trees when k is the size of the alphabet. In this paper, we present an algorithm that integrates the multiple parse trees into one and emulates encoding and decoding on it. We also present that we can reduce the total number of nodes which must be constructed, by $\Omega(k^2)$.

1. はじめに

テキスト圧縮は，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．これは主に，歪のない情報源符号化を用いて実現される．

†1 北海道大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Hokkaido University

今日までに，Huffman 符号や Run-length 法，LZ 系圧縮法など，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[3),5)]．

歪のない情報源符号化において，符号とは，情報源アルファベット上の任意の記号列を符号アルファベット上の記号列に写す 1 対 1 写像に他ならない．このような観点からみると，符号化手法は次の 4 種類に大別できる．

FF 符号 (fixed-length-to-fixed-length code): ある一定の長さ L ごとに情報源系列を分割し，それぞれを固定長 ℓ の符号語へと変換する符号化．

FV 符号 (fixed-length-to-variable-length code): ある一定の長さ L ごとに情報源系列を分割し，それぞれを可変長の符号語へと変換する符号化．

VF 符号 (variable-length-to-fixed-length code): 可変の長さに情報源系列を分割し，それぞれを固定長 ℓ の符号語へと変換する符号化．

VV 符号 (variable-length-to-variable-length code): 可変の長さに情報源系列を分割し，それぞれを可変長の符号語へと変換する符号化．

良く知られた Huffman 符号は，固定長の記号列（1 文字ごと）に可変長の符号語を割り当てる FV 符号である．また，LZ 系圧縮法などは VV 符号とみることができる．

多くの場合において，テキスト圧縮で重要視される要素は圧縮率である．そのため，現在，可変長符号化である FV 符号や VV 符号の研究が主流である[4)]．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 符号が見直されている[1),2)]．

代表的な VF 符号である Tunstall 符号[6)] 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 1 文字あたりの平均符号長が漸近する．しかしながら，実際のところ，その漸近する速度は符号語長に対して非常に緩やかであり，現実的には Huffman 符号ほどの圧縮率を得ることは難しい[8)]．

VF 符号の欠点である圧縮率の低さを改善するため，Tunstall 符号を元に，Yamamoto と Yokoo らは AIVF 符号 (Almost Instantaneous VF code) を提案した[7)]．Tunstall 符号において各符号語は分節木の葉に割り当てられるが，AIVF 符号では，分節木の内部ノードにも符号語を割り当てることで無駄な枝を刈り込み，逆に平均符号長を増加させる有益な枝を伸長する．そのような符号化では，情報源記号上の語頭条件を満たさず，復号時に問題があるように見えるが，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．また，そのようにして分割した情報源系列（ブロックと呼ばれる）の前

Vol.2009-DBS-148 No.21
Vol.2009-FI-95 No.21
2009/7/28

後は，元がたとえ無記憶情報源の系列であったとしても，独立ではなくなり文脈が生じる．この性質を利用し，Yamamoto と Yokoo らは AIVF 符号において，複数の分節木を文脈によって切り替えることで辞書を拡大する手法も示している[7]．実際，複数の分節木を用いた AIVF 符号（多重分節木による AIVF 符号）は符号語長を短くとっても，十分長い符号語長の Tunstall 符号より圧縮率が良い．

しかしながら，多重分節木による AIVF 符号の符号化・復号化の速度は非常に遅い．なぜなら，情報源アルファベットのサイズを k とすると，$k-1$ 個もの分節木を構築しなければならず，自然言語のテキストのように k が 70〜140 にもなる場合には，単一の分節木を使用する VF 符号より圧倒的に時間がかかる．また，当然ながら，それらの多重分節木を保持する領域もより多くかかる．VF 符号で符号化されたテキストに対してパターン照合を行う場合，明示的に符号語列を情報源系列へと復号はしないが，符号化時に用いた符号木を復元する必要がある[9] ため，分節木の構築に時間がかかることは，パターン照合処理を高速化する上でのボトルネックになる．また，符号木にパターン照合のための付加情報を計算する必要もあるので，木はなるべくコンパクトであることが望ましい．したがって，これらの問題を解決するためには，多重分節木による AIVF 符号と同等の圧縮率を達成しつつも，よりコンパクトな分節木で符号化を実現できる手法が望ましい．

本稿では，Yamamoto と Yokoo らの多重分節木による AIVF 符号（以降，特にこれを YY 符号と呼ぶことにする）における複数の分節木を統合し，1 個の大きな分節木としてまとめ，その木の上で多重分節木を模倣して YY 符号と同等の符号化を行うアルゴリズムを提案する．また，これにより，構築すべき分節木に含まれる総ノード数を，情報源アルファベットサイズ k に対して $\Omega(k^2)$ 個削減できることを示す．

2. 準　　備

2.1 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．特に，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．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．

任意の文字列 $x \in \Sigma^*$ のテキスト S 中の出現確率を $\Pr_S(x)$ と書く．また，便宜上 $\Pr_S(\varepsilon) = 1$ と定義する．当然 $\Pr_S(x)$ は S に依存するが，文脈から対象のテキストが明らかな場合や，情報源の統計的性質として取り扱う場合には，単に $\Pr(x)$ と書く．

木構造のうち，各ノードの子の数が高々k 個の木を k 分木と呼ぶ．また，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）と呼ぶ．親を持たないノード（すなわち木の頂点）をルートノードあるいは単にルートと呼ぶ．さらに，k 分木において，子の数が k である内部ノードを完全内部ノード，子の数が k 未満である内部ノードを不完全内部ノードと呼ぶ．木（または森）T について，T に含まれる葉ノード全体の集合を $\mathcal{L}(T)$ と書く．また，葉ノード全体と不完全内部ノード全体を合わせた集合を $\mathcal{N}(T)$ と書く．集合 S の大きさは $\#S$ で表す．よって，たとえば，T 中の葉ノードの総数は，$\#\mathcal{L}(T)$ と書ける．すべての内部ノードが完全内部ノードであるものを，完全 k 分木と呼ぶ．あるノード n の子の数を度数とよび，$d(n)$ と書く．

2.2 AIVF 符号

Tunstall 符号[6] は，完全 k 分木を分節木として使用する VF 符号化である．したがって，Tunstall 符号の符号木 T^* で 2 元符号化する際には，$\#\mathcal{L}(T^*) = 2^l$ となる整数 l が存在しなければ，未使用の符号語が発生することになる．このことは，その未使用の符号語に他のブロックを割り当てれば，平均ブロック長を長くすることができることを示唆している．AIVF 符号[7] は，そのような完全ではない k 分木を分節木として用いることで圧縮率を向上させている．すなわち，AIVF 符号とは，分節木のすべての葉および不完全内部ノードに符号語が割り当てられ，完全内部ノードには符号語が割り当てられていない分節木を用いる VF 符号のことである．

以下に単一の分節木による AIVF 符号の手順について説明する．なお，$\Sigma = \{a_1, \ldots, a_k\}$ とし，便宜上，Σ の各記号は出現確率順でソートされているものとする．すなわち，$i < j$ ならば $\Pr(a_i) \geq \Pr(a_j)$ であると仮定する．

AIVF 符号化は，構築された分節木 T のルートをスタート地点とし，入力テキストから 1 文字ずつ読み込みながら分節木を探索する．探索中に，子をそれ以上たどれなくなったらルートへと戻り，戻る際に到達したノードに割り当てられた符号語を出力する．その手順は，アルゴリズム 1 のとおりである．

例 1　図 1 の分節木を用いてテキスト $S = aabaabaccab$ を符号化する場合を考える．S は分節木により，$aa \cdot ba \cdot ab \cdot ac \cdot c \cdot ab$ と分割される．各ブロックを符号化すると，$001 \cdot 100 \cdot 010 \cdot 011 \cdot 110 \cdot 010$ が出力される．

Vol.2009-DBS-148 No.21
Vol.2009-FI-95 No.21
2009/7/28

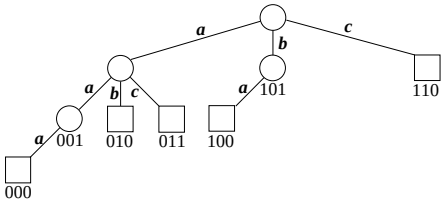

図 1 AIVF 符号化の分節木

アルゴリズム 1 AIVF 符号化

$n \leftarrow T$ のルートノード
while 入力テキストの最後ではない **do**
  $c \leftarrow$ 入力テキストの次の記号
  if n に c でラベル付けされた子が存在しない **then**
    n に割り当てられた符号語を出力
    $n \leftarrow T$ のルートノード
  end if
  $n \leftarrow n$ の c でラベル付けされた子
end while

AIVF 符号の分節木 T により符号語が割り当てられるブロック全体の集合を C_T とする．すなわち，C_T は T に登録される文字列全体からなる辞書といえる．今，符号語の個数が M，すなわち $\#C_T = \#\mathcal{N}(T) = M$ である分節木のうち，平均ブロック長 $\sum_{x \in C_T} |x| \cdot \Pr(x)$ を最長にする分節木 T^* は，アルゴリズム 2 によって構築される[7]．

アルゴリズム 2 は，基本的には平均ブロック長を最長化するノードを作成し続ける．ただし，あるノードから $k-1$ 番目の子が追加されたとすると，そのノードに符号語を割り当てるより，k 番目の子を作って符号語を割り当てたほうが平均ブロック長を長くすることができることに注意する[⋆1]．これによる不規則な平均ブロック長の増加を **while** ループ内で反映している．また，**while** ループ終了後には，符号語を付けるノードの数が M に満たない場合がある．このときには，符号語を付けるノードの数が M となるまで，平均ブロック長を最長化するノードを作成し続ける．

⋆1 k 番目の子を作れば，そのノードは完全内部ノードとなり，符号語を割り当てる必要がなくなるからである．

アルゴリズム 2 AIVF 符号の最適な分節木の構築

ルートノード n_0 とその子 n_j $(j = 1, \ldots, k)$ からなる木 T^* を作成する．
$m \leftarrow k$.
$\hat{n} \leftarrow \underset{n \in \mathcal{N}(T^*)}{\operatorname{argmax}} \{\Pr(n)\}$.
while $k - d(\hat{n}) - 1 \leq M - m$ **do**
  $S_1 \leftarrow$ アルゴリズム 3 を実行したときの平均ブロック長.
  $S_2 \leftarrow$ アルゴリズム 4 を $k - d(\hat{n}) - 1$ 回実行したときの平均ブロック長.
  if $S_1 \geq S_2$ **then**
    アルゴリズム 3 を実行する．
  else
    アルゴリズム 4 を $k - d(\hat{n}) - 1$ 回実行する．
  end if
  $\hat{n} \leftarrow \underset{n \in \mathcal{N}(T^*)}{\operatorname{argmax}} \Pr(n)$
end while
アルゴリズム 4 を $M - m$ 回実行する
それぞれのノードの j 番目の辺を a_j $(j = 1, \ldots, k)$ でラベル付けをする．
葉と不完全ノードに符号語 m $(m = 0, \ldots, M-1)$ を割り当てる．

アルゴリズム 3

$\hat{n} \leftarrow \underset{n \in \mathcal{N}(T^*)}{\operatorname{argmax}} \Pr(n)$
$\hat{n}$ に，$k - d(\hat{n})$ 個の子ノード n_j $(j = d(\hat{n}) + 1, \ldots, k)$ を付け足す．

アルゴリズム 4

$\left(\tilde{n}, \tilde{j}\right) \leftarrow \underset{n \in \mathcal{N}(T^*),\, j=1,\ldots,k}{\operatorname{argmax}} \Pr(n) \Pr(a_j)$
$\tilde{n}$ に，$\tilde{j}$ 番目の子ノード n を付け足す．

Vol.2009-DBS-148 No.21
Vol.2009-FI-95 No.21
2009/7/28

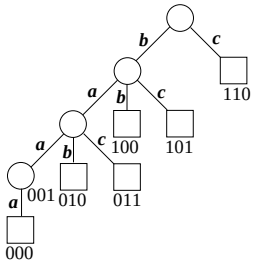

図 2 もう 1 つの分節木

例 2 $\Pr(a) = 0.6$ および，$\Pr(b) = 0.3$, $\Pr(c) = 0.1$ である情報源に対して，$M = 7$ でアルゴリズム 2 を適用すると，図 1 に示される分節木が得られる．

2.3 多重分節木による AIVF 符号 (YY 符号)

前節の単一の分節木による AIVF 符号では，たとえ入力テキストが記憶のない情報源であったとしても，ブロックが必ずしも統計的に独立しているとは限らない．例えば，図 1 の分節木で符号化しているときに，ブロックが aa となり，001 が出力されたとする．このときに，次の文字は b か c でなければならない．なぜならば，もし次の文字が a だとすれば，ブロックが aaa となり 000 が出力されるからである．このように，001 が出力された場合には，図 1 の分節木の $000, 001, 010, 011$ のノードには決してたどりつくことはない．このことは，たどりつくことのないノードの符号語に，別のブロックを割り当てることによって，平均ブロック長を長くできることを示唆している．

YY 符号[7] では，それぞれルートが $a_{i+1}, \ldots, a_k$ $(i = 0, \ldots, k-2)$ でラベル付けされた子を持つ $k-1$ 個の分節木 T_i $(i = 0, \ldots, k-2)$ を作成しておき，それらを切り替えながら使用して符号化する．ここで，使用できる符号語の個数を M とすると，各 T_i について，$\#\mathcal{N}(T_i) = M$ となることが，YY 符号の優れた圧縮率を導いている．そのような多重分節木を用いた符号化は，アルゴリズム 5 のようにして行われる．

例 3 図 1 と図 2 の分節木を用いてテキスト $S = aabaabaccab$ を符号化するとき，S は $aa \cdot baa \cdot bac \cdot c \cdot ab$ と分割され，$001 \cdot 001 \cdot 011 \cdot 110 \cdot 010$ と符号化される．

平均ブロック長を最長にする多重分節木 T_i $(i = 0, \ldots, k-2)$ を作成する方法は，前節の分節木の作成方法とほぼ同じである．ただし，前節では，最初にルートノードから k 個の子を作成していたが，ここでは，ルートノードから $k-i$ 個の子を作成する．このような多重分節木は，アルゴリズム 6 によって構築される．

アルゴリズム 5 多重分節木による符号化

```
1: T ← T_0
2: n ← T_0 のルートノード
3: while 入力テキストの最後ではない do
4:    c ← 入力テキストの次の記号
5:    if n に c でラベル付けされた子が存在しない then
6:       n に割り当てられた符号語を出力
7:       T ← T_{d(n)}
8:       n ← T のルートノード
9:    end if
10:   n ← n の c でラベル付けされた子
11: end while
```

アルゴリズム 6 多重分節木 T_i の構築

```
1: for i = 0 to k − 2 do
2:    ルートノード n_0 とその k − i 個の子 n_j (j = 1, ... k − i) を作成する．
3:    m ← k − i
4:    アルゴリズム 2 の 3 行目〜14 行目を実行する．
5:    ルートの j 番目の枝を a_{i+j} (j = 1, ..., k − i) でラベル付けをする．
6:    ルート以外のノードの j 番目の枝を a_j (j = 1, ..., k) でラベル付けする．
7:    葉と不完全内部ノードに符号語 m (m = 0, ..., M − 1) を割り当てる．
8: end for
```

例 4 $\Pr(a) = 0.6$ および，$\Pr(b) = 0.3$, $\Pr(c) = 0.1$ である情報源に対して，$M = 7$ でアルゴリズム 6 を適用すると，T_0 として，図 1 に示される分節木が得られ，T_1 として，図 2 に示される分節木が得られる．

3. Virtual Multiple-AIVF 木 (VMA 木)

節 2.3 の YY 符号は，$k-1$ 個の分節木を構築して保持するため，符号化・復号化時に多

Vol.2009-DBS-148 No.21
Vol.2009-FI-95 No.21
2009/7/28

くの時間と領域がかかる．そこで，多重分節木の部分木間でノードを共有することで，総ノード数を削減することを考える．簡単な観察から，T_i のルートから a_{i+1} でたどった子を頂点とする部分木（T_i の一番左の部分木）以外のノードは，T_{i+1} の木に完全に含まれているように見える．まずは，この関係を証明しよう．

今，T_i のルートの a_j でラベル付けされた子を頂点とする部分木を $S_j^{(i)}$ で表すとする．このとき，次の定理が成り立つ．

定理 1　任意の整数 i $(0 \le i \le k-3)$ と j $(2 \le j \le k-i)$ について，$S_{i+j}^{(i)}$ は，$S_{i+j}^{(i+1)}$ に完全に含まれる．

証明．多重分節木 T_i $(i = 0, \ldots, k-2)$ のルートには，a_j $(j = i+1, \ldots, k)$ でラベル付けされた子が存在する．つまり，T_i は，$S_j^{(i)}$ $(j = i+1, \ldots, k)$ が含まれている．任意の i $(0 \le i \le k-2)$ について，ルートのみ $a_{i+1}, \ldots, a_k$ でラベル付けられた子を持ち，それ以外はすべて k 個の子を持つような，大きさ無限大の木を T_i^∞ とする．各 T_i は，その構成方法から，平均ブロック長が最長であるような木であることに注意しよう[7]．すなわち，任意の整数 i $(0 \le i \le k-3)$ と j $(2 \le j \le k-i)$ について，$S_{i+j}^{(i)}$ 中の任意のノードは，T_i^∞ に含まれ，かつ T_i には含まれない任意のノードと交換[⋆1]しても，平均ブロック長を長くすることはできないことを意味している．ここで，$S_{i+j}^{(i)}$ が $S_{i+j}^{(i+1)}$ に含まれていないと仮定する．すると，$S_{i+j}^{(i)}$ に含まれていて，$S_{i+j}^{(i+1)}$ には含まれていないノードが存在する．そのようなノードを n とする．今，T_i は平均ブロック長が最長である木なので，$S_{i+j}^{(i+1)}$ に含まれて，かつ $S_{i+j}^{(i)}$ に含まれていないノードと n を交換すれば，T_{i+1} の平均ブロック長を長くすることができる．このことは，T_{i+1} が平均ブロック長が最長となる分節木であることに矛盾する．したがって，$S_{i+j}^{(i)}$ は，$S_{i+j}^{(i+1)}$ に完全に含まれる．　□

上述の定理より，各 T_i を重ね合わせて一つの統合した木として保持できることが観察できる．この統合した木を Virtual Multiple-AIVF 木（以降 VMA 木と省略する）と呼ぶことにする．ただし，各ノードが元のどの分節木 T_i に含まれていたかについて，別途印をつけておく必要がある．これは，各ノード n について，n が元々属している木 T_i のうち最小の i を保持することで実現できる．この印を $\mathrm{Tn}(n)$ と書くとすると，すなわち，$\mathrm{Tn}(n) = \min_i\{i \mid 0 \le i \le k-2, n \text{ は } T_i \text{ に含まれる.}\}$ である．

例 5　図 1 の木と図 2 の木とを重ね合わせると，図 3 の木が得られる．

⋆1 ノード a とノード b とを交換するとは，分節木からノード a を削除して，ノード b を付け加えることをいう．

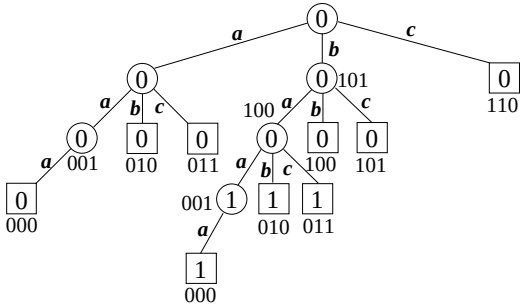

図 3　図 1 の木と図 2 の木とを重ね合わせた木
各ノード n の中に書かれている数字は，$\mathrm{Tn}(n)$ である．

アルゴリズム 7 VMA 木による符号化

```
1: n ← T のルートノード
2: i ← 0
3: while 入力テキストの最後ではない do
4:    c ← 入力テキストの次の記号
5:    if n に c でラベル付けされた子が存在する then
6:       n′ ← n の c でラベル付けされた子
7:       if Tn(n′) ≤ i then
8:          n ← n′
9:       else
10:         13 行目にジャンプ
11:      end if
12:   else
13:      w_i(n) を出力
14:      i ← d(n)
15:      n ← T のルートノード
16:      n ← n の c でラベル付けされた子
17:   end if
18: end while
```

Vol.2009-DBS-148 No.21
Vol.2009-FI-95 No.21
2009/7/28

アルゴリズム 8 VMA 木の構築アルゴリズム

$T \leftarrow$ ルートノードから k 個の子を作った木
T の j 番目の辺を a_j でラベル付けする．
for $l = 0$ to $k-2$ **do**
  $m \leftarrow \#\mathcal{N}(S_l)$
  $\hat{n} \leftarrow \underset{n \in \mathcal{N}(T)}{\operatorname{argmax}} \Pr(n)$
  while $k - d(\hat{n}) - 1 \leq m$ **do**
    $S_1 \leftarrow$ アルゴリズム 9 を実行したときの平均ブロック長
    $S_2 \leftarrow$ アルゴリズム 10 を $k - d(\hat{n}) - 1$ 回実行したときの平均ブロック長
    if $S_1 \geq S_2$ **then**
      アルゴリズム 9 を実行する
    else
      アルゴリズム 10 を $k - d(\hat{n}) - 1$ 回実行する
    end if
    $\hat{n} \leftarrow \underset{n \in \mathcal{N}(T)}{\operatorname{argmax}} \Pr(n)$
    $m \leftarrow m - k + d(\hat{n}) + 1$
  end while
  アルゴリズム 10 を m 回実行する
  $i \leftarrow 0$
  for all $n \in \mathcal{N}(D)$ **do**
    $w_l(n) \leftarrow i$
    $i \leftarrow i + 1$
  end for
  ルートから a_{l+1} をたどったノードを頂点とする部分木を S とする．
  T の S 以外の部分木を R とおく．
  T_{V} に S を付け足す
  $T \leftarrow R$
end for
全てのノードの j 番目の辺を a_j $(j = 1, \ldots, k)$ でラベル付けする．

アルゴリズム 9

$\hat{n} \leftarrow \underset{n \in \mathcal{N}(T)}{\operatorname{argmax}} \Pr(n)$
$\hat{n}$ の下に $k - d(\hat{n})$ 個の子ノード n_j, $j = d(\hat{n}) + 1, \ldots, k$ を付け足す．
$\mathrm{Tn}(n_j) \leftarrow \mathrm{cod}(\mathrm{first}(n_j)) - 1$

アルゴリズム 10

$\left(\tilde{n}, \tilde{j}\right) \leftarrow \underset{n \in \mathcal{N}(T),\, j=1,\ldots,k}{\operatorname{argmax}} \Pr(n) \Pr(a_j)$
$\tilde{n}$ の下に $\tilde{j}$ 番目の子ノード n を付け足す．
$\mathrm{Tn}(n_j) \leftarrow \mathrm{cod}(\mathrm{first}(n)) - 1$

この VMA 木を用いて符号化を行うには，アルゴリズム 5 はそのまま使用することはできない．なぜなら，現在のノードから入力テキストの次の記号でラベル付けされた子が存在していても，それが現在探索中の T_i に存在するとは限らないからである．そこで，現在探索中の木 T_i の番号 i と $\mathrm{Tn}(n)$ を比較し，i の方が小さければ符号語を出力してルートへ戻るようにする．この符号化の手順はアルゴリズム 7 のとおりである．ただし，T_i のノード n に割り当てられた符号語を $w_i(n)$ で表す．

また，VMA 木 T_{V} の具体的な構築アルゴリズムは，アルゴリズム 8 のとおりである．ここで，ルートの a_i でラベル付けされた子を頂点とする部分木を S_i で表す．便宜上，$\#\mathcal{N}(S_0) = M$ とする．また，$\mathrm{cod}(a_j) = j$ と定義し，ルートからノード n へ至る辺のラベルを並べた文字列の先頭を $\mathrm{first}(n)$ で表す．すなわち，ルートからノード n へ至るラベル列の先頭文字が a_j であった場合，$\mathrm{cod}(\mathrm{first}(n)) = j$ となる．これまでのアルゴリズムと違い，このアルゴリズムでは，m をこれから作成する符号語を持つノードの数としている．

4. 削減されるノード数についての考察

使用できる符号語の数が M である場合について考える．YY 符号における i 番目の分節木 T_i の完全内部ノードの数を m_i^{C} と書く．また，多重分節木全体のノード数を N と書く．このとき，各 T_i には，符号語が割り当てられるノードが $\#\mathcal{N}(T_i) = M$ 個存在するので，

Vol.2009-DBS-148 No.21
Vol.2009-FI-95 No.21
2009/7/28

次式が成り立つ．

$$N = M(k-1) + \sum_{i=0}^{k-2} m_i^{\mathsf{C}}. \tag{1}$$

一方，VMA 木 T_{V} の全ノード数を N_{V} と書く．また，T_{V} の完全内部ノードの数を $m_{\mathsf{V}}^{\mathsf{C}}$ と書く．このとき，明らかに $N_{\mathsf{V}} = \#\mathcal{N}(T_{\mathsf{V}}) + m_{\mathsf{V}}^{\mathsf{C}}$ が成り立つ．すると，$\#\mathcal{N}(T_{\mathsf{V}}) = \sum_{i=1}^{k} \#\mathcal{N}(S_i)$ であるが，アルゴリズムの構成方法から，かならず $\#\mathcal{N}(S_{k-1}) + \#\mathcal{N}(S_k) = M$ となるので，$\#\mathcal{N}(T_{\mathsf{V}}) = M + \sum_{i=1}^{k-2} \#\mathcal{N}(S_i)$ となる．したがって，

$$N_{\mathsf{V}} = M + \sum_{i=1}^{k-2} \#\mathcal{N}(S_i) + m_{\mathsf{V}}^{\mathsf{C}} \tag{2}$$

となる．また，S_i は，多重分節木の木 T_{i-1} の最も左側の部分木 $S_i^{(i-1)}$ に等しく，それより右側にある部分木（$k-i$ 個ある）にはそれぞれ少なくとも 1 個の符号語が割り当てられることから，任意の $1 \le i \le k$ について，

$$\#\mathcal{N}(S_i) \le M - (k-i) \tag{3}$$

が成り立つことに注意する．

したがって，削減されるノード数 $N - N_{\mathsf{V}}$ について，以下の式が成り立つ．

$$\begin{aligned} N - N_{\mathsf{V}} &= \left\{ M(k-1) + \sum_{i=0}^{k-2} m_i^{\mathsf{C}} \right\} - \left\{ M + \sum_{i=1}^{k-2} \#\mathcal{N}(S_i) + m_{\mathsf{V}}^{\mathsf{C}} \right\} \quad \text{(式 1，2 より)} \\ &\ge \left\{ M(k-1) + \sum_{i=0}^{k-2} m_i^{\mathsf{C}} \right\} - \left\{ M + \sum_{i=1}^{k-2} (M - (k-i)) + m_{\mathsf{V}}^{\mathsf{C}} \right\} \quad \text{(式 3 より)} \\ &= \left(\sum_{i=0}^{k-2} m_i^{\mathsf{C}} - m_{\mathsf{V}}^{\mathsf{C}} \right) + \left(\frac{k^2}{2} - \frac{k}{2} - 1 \right). \end{aligned}$$

VMA 木 T_{V} に含まれる任意の完全内部ノード n^{C} について，明らかに n^{C} は多重分節木のある木 T_i に完全に含まれているので，この式の最初の括弧部分 $\left(\sum_{i=0}^{k-2} m_i^{\mathsf{C}} - m_{\mathsf{V}}^{\mathsf{C}} \right)$ は 0 以上である．$\Delta = \left(\sum_{i=0}^{k-2} m_i^{\mathsf{C}} - m_{\mathsf{V}}^{\mathsf{C}} \right)$ と置くと，結局のところ，

$$N - N_{\mathsf{V}} \ge \left(\frac{k^2}{2} - \frac{k}{2} - 1 \right) + \Delta \quad (\Delta \ge 0)$$

と書ける．したがって，以上の議論から，削減されるノード数は $\Omega(k^2)$ であることが分かる．

5. おわりに

本稿では，多重分節木による AIVF 符号を，統合した一つの木によって模倣するアルゴリズムを提案した．これによって，分節木全体として $\Omega(k^2)$ 個のノード数を削減できることを示した．このことは，多重分節木による AIVF 符号による符号化・復号化時において，時間・領域コストの双方を $\Omega(k^2)$ 削減できることを意味している．また，符号上での圧縮パターン照合の観点からも，前処理のコストが削減できる．実際のテキストに対してどのくらいの効果があるのかについての実証実験は，今後の課題である．

参 考 文 献

1) Kida, T.: Suffix Tree Based VF-Coding for Compressed Pattern Matching, *Proc. of Data Compression Conference 2009(DCC2009)*, p.449 (2009).
2) Klein, S.T. and Shapira, D.: Improved Variable-to-Fixed Length Codes, *SPIRE '08: Proceedings of the 15th International Symposium on String Processing and Information Retrieval*, Berlin, Heidelberg, Springer-Verlag, pp.39–50 (2009).
3) Salomon, D.: *Data Compression: The Complete Reference*, Springer, 4th edition (2006).
4) Salomon, D.: *Variable-length Codes for Data Compression*, Springer (2007).
5) Sayood, K.(ed.): *Lossless Compression Handbook*, Academic Press (2002).
6) Tunstall, B.P.: Synthesis of noiseless compression codes, PhD Thesis, Georgia Inst. Technol., Atlanta, GA (1967).
7) Yamamoto, H. and Yokoo, H.: Average-Sense Optimality and Competitive Optimality for Almost Instantaneous VF Codes, *IEEE Trans. on Information Theory*, Vol.47, No.6, pp.2174–2184 (2001).
8) 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*, Vol.8, No.1, p. (2009). (to appear).
9) 喜田拓也：VF 符号上における圧縮照合アルゴリズム，技術報告，電子情報通信学会コンピュテーション研究会（COMP）(2009).