ERATO セミナ 2010 - No. 07

効率よい VF 符号の設計

喜田 拓也
北海道大学大学院 情報科学研究科

2010/7/23

概要

VF 符号とは、情報源系列を可変長のブロックに分割し、各ブロックに対して固定長の符号語を割り当てることで圧縮を達成する情報源符号化の一種である。入力系列（テキスト）の分割には、主に分節木と呼ばれる木構造が用いられる。VF 符号は、近年、パターン照合を高速化することのできるデータ圧縮法として見直されている。しかしながら、VF 符号は、符号語長が固定であるという制約から高い圧縮率を得ることが難しく、実用的な符号化方法は存在しなかった。発表者らは、これまで、テキストに対する接尾辞木をもとに、それを刈り込んだ木を分節木とすることで圧縮率の高い符号化を提案し、その改良を行ってきた。本発表では、我々の手法とその実験的評価について紹介する。

北海道大学 Hokkaido University

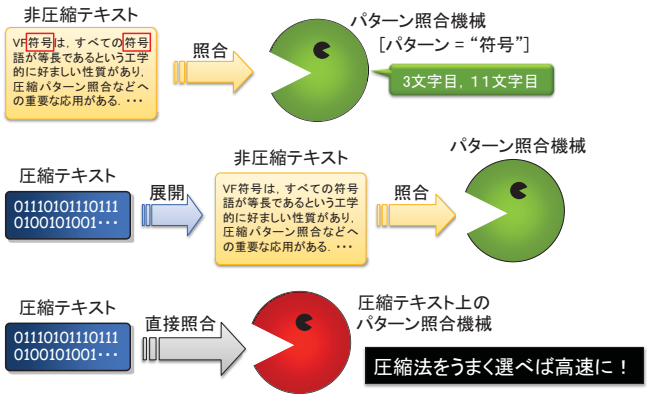
効率よいVF符号の設計

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
喜田拓也

発表概要

- 研究背景と目的
  - 圧縮パターン照合
  - 圧縮パターン照合に適した圧縮法
- VF符号について
  - Tunstall符号
- STVF符号
  - オリジナル版
  - Almost instantaneousによる改善
  - 分節木の繰り返し学習による改善
- まとめ

研究の背景：圧縮パターン照合問題(CPM問題)



研究の背景：CPM問題に適したデータ圧縮

- 圧縮パターン照合に適した圧縮法の特徴
  - 符号語の区切りが明確である
    - 特にバイト単位や固定長の符号語であることが望ましい
  - 静的でコンパクトな辞書を用いる
    - 動的に辞書が変化すると，テキストスキャン時に余計な時間がかかる
  - (その上でなるべく)圧縮率が良いこと
    - 処理すべきデータ量が削減される

VF符号
(Variable-length-to-Fixed-length Code)

- Tunstall符号(Tunstall 1967)
- AIVF符号(Yamamoto&Yokoo 2001)
- STVF符号(Klien&Shapira2008, Kida2009)

関連研究

- U. Manber: A text compression scheme that allows fast searching directly in the compressed file. In *Proc. Combinatorial Pattern Matching*, LNCS807, pages 113–124. Springer–Verlag, 1994.
- P. Gage: A new algorithm for data compression. *The C Users Journal*, 12(2), 1994. [BPE]
- E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza–Yates. Direct pattern matching on compressed text. In *Proc. 5th International Symp. on String Processing and Information Retrieval*, pages 90–95. IEEE Computer Society, 1998. [Word-based Huffman]
- E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza–Yates. Fast and flexible word serching on compressed text. ACM Trans. On Information Systems 18, 113–139, 2000. [Tagged Huffman]
- 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, 2001. [AIVF]
- Brisaboa, N.R., Iglesias, E.L., Navarro, G., Parama, J.R.: An efficient compression code for text databases. *In: ECIR*. pp. 468–481, 2003. [ETDC]
- Brisaboa, N.R., Farina, A., Navarro, G., Esteller, M.F.: (s, c)–dense coding: An optimized compression code for natural language text databases. *In: SPIRE*. pp. 122–136, 2003. [SCDC]
- Klein, S.T., Kopel Ben–Nissan, M.: Using Fibonacci compression codes as alternatives to dense codes. In: Proc. DCC2008, pp.472–481, 2008. [Word-based Fibonacci coding]
- Maruyama, S., Tanaka, Y., Sakamoto, H., Takeda, M.: Context–sensitive grammar transform: Compression and pattern matching. *In: SPIRE2008*. LNCS5280, pp. 27–38. [BPEX]
- Klein, S.T., Shapira, D.: Improved variable–to–fixed length codes. In: proc. SPIRE2008, pp. 39—50, 2009. [DynC]
- Brisaboa, N.R., Farina, A., Lopez, J.R., Navarro, G., Lopez, E.R.: A new searchable variable–to–variable compressor. In: DCC. pp. 199—208, 2010.

VF符号について

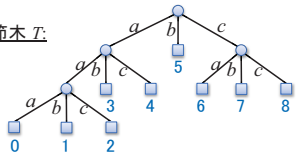
VF符号について

		圧縮テキスト(符号語) 固定長	圧縮テキスト(符号語) 可変長 (ハフマン符号など)
入力テキスト(情報源記号)	固定長	**FF符号** (Fixed length to Fixed length code)	**FV符号** (Fixed length to Variable length code)
	可変長	**VF符号** (Variable length to Fixed length code) Tunstall符号	**VV符号** (Variable length to Variable length code) LZ系など

- 今の主流の圧縮法は，ほぼVV符号
- VF符号は符号語が固定長で，圧縮率を上げにくい
  - 既存の手法で実用的に使われているものは皆無

分節木を用いたVF符号

- 入力テキストを，分節木と呼ばれる木構造を用いて可変長のブロックに分割し，それぞれに固定長の符号化を割り当てることで符号化を行う
  - 分節木の各ノードは文字列に対応している
  - 文字列が辞書(分節木)に登録されており，それぞれに固定長の符号語が割り当てられていると考えて良い



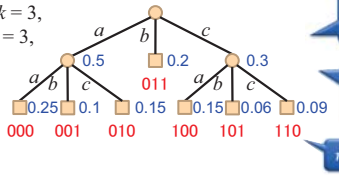
入力テキスト:	a	b	b	a	a	b	b	a	a	a	c	a	c	c
符号化テキスト:	3		5		1		5		0		6		8	

Tunstall符号

B. P. Tunstall, "Synthesis of noiseless compression codes," Ph.D. dissertation, Georgia Ins. Technol., Atlanta, GA, 1967

- 情報源アルファベット $\sum$ ($|\sum| = k$)に対して，完全 k-分木を分節木として用いるVF符号．各文字の生起確率が既知である記憶のない情報源に対して(平均ブロック長を最長にするという意味で)最適なVF符号

Tunstall木 T_m:
$\sum = \{a, b, c\}, k = 3,$
内部接点数 $m = 3,$
$P(a) = 0.5,$
$P(b) = 0.2,$
$P(c) = 0.3,$



葉の番号を固定長の2進数表現したもの

符号長3で符号化
符号語の数の上限は $2^3 = 8$個

$m = \lfloor (2^\ell - 1)/(k-1) \rfloor$

例)情報源系列 S = $abacaacbabcc$ を上のT_mを使って符号化すると，

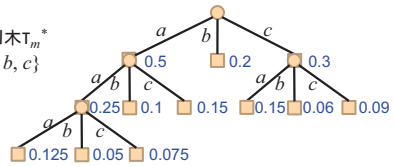
S = $ab \cdot ac \cdot aa \cdot cb \cdot ab \cdot cc$

Z = 001 010 000 101 001 110

Tunstall木の構築

- 平均ブロック長を最大にする最適な文節木T_m^*の構築
  - 与えられた $m \geqq 1$ と各アルファベット記号の出現確率 $P(a)$ $(a \in \sum)$に対し，次のステップにより文節木T_m^*を構築する
    1. 高さが 1 の k-文木 T_1 を初期の文節木 T_1^* とする．
    2. 次のステップを各 $i = 2, \ldots, m,$ について繰り返す
       - A) 木T_{i-1}^*のすべての葉の中から，最大の確率を持つノード $v = v_i^*$ を選ぶ
       - B) 初期木 T_1 を v_i^* の下に接ぎ木して T_i^* を作る

Ex) Tunstall木T_m^*
$\sum = \{a, b, c\}$
$k = 3,$
$m = 4$



$P(a) = 0.5$
$P(b) = 0.2$
$P(c) = 0.3$

STVF符号

Suffix Tree based VF Codes

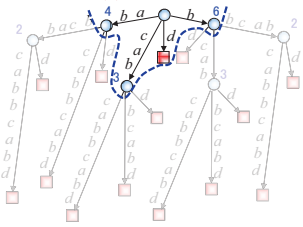
圧縮率改善のアイデア

- 与えられたテキストに対する接尾辞木を分節木に利用する
  - 接尾辞木(suffix tree)[Weiner, 1973]はテキストの完全な統計情報を持つ
- テキストT に対する接尾辞木$ST(T)$は，T の任意の部分文字列に対する辞書木になっている
  → そのままでは分節木として用いることができない！
  - 適切に木を刈り込んで，分節木を構築する必要がある
  - なるべく頻度の高いノードを残すように枝を刈る
- 辞書となる刈り込み接尾辞木は，圧縮テキストを展開するときにも必要となるので，圧縮データと共に保存する
- DCC2009でポスター発表するも，類似先行研究あり！
  - Klein, S.T., and Shapira, D, "Improved variable-to-fixed length codes," In : SPIRE2008, pp. 39-50, 2008.

刈り込み接尾辞木による文節木

- 接尾辞木
  - 入力テキストのすべての接尾辞が登録された圧縮トライ
  - 長さ n のテキストに対し，木のサイズは $O(n)$
  - 接尾辞木を用いると，任意の部分文字列が検索可能
  - $O(n)$ 時間でのオンライン構築アルゴリズムが存在
    - E. Ukkonen, Constructing suffix trees on-line in linear time, *Proc. of IFIP'92*, pp.484-492.

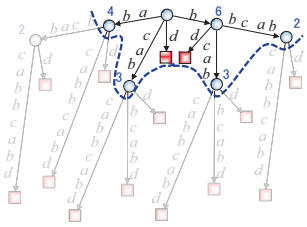
接尾辞木 $ST(S)$: S=abbcabcabbcabd



刈り込み接尾辞木による文節木

- 接尾辞木
  - 入力テキストのすべての接尾辞が登録された圧縮トライ
  - 長さ n のテキストに対し，木のサイズは $O(n)$
  - 接尾辞木を用いると，任意の部分文字列が検索可能
  - $O(n)$ 時間でのオンライン構築アルゴリズムが存在
    - E. Ukkonen, Constructing suffix trees on-line in linear time, *Proc. of IFIP'92*, pp.484-492.

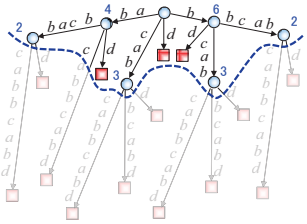
接尾辞木 $ST(S)$: S=abbcabcabbcabd



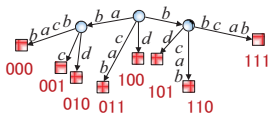
刈り込み接尾辞木による文節木

- 接尾辞木
  - 入力テキストのすべての接尾辞が登録された圧縮トライ
  - 長さ n のテキストに対し，木のサイズは $O(n)$
  - 接尾辞木を用いると，任意の部分文字列が検索可能
  - $O(n)$ 時間でのオンライン構築アルゴリズムが存在
    - E. Ukkonen, Constructing suffix trees on-line in linear time, *Proc. of IFIP'92*, pp.484-492.

接尾辞木 $ST(S)$: S=abbcabcabbcabd



文節木 $PT(S)$



$$S = abbcab \cdot cab \cdot bcab \cdot d$$
$$000 \quad 011 \quad 110 \quad 100$$

（頻度優先）刈り込みアルゴリズム

1. 接尾辞木$ST(T\$)$を構築する．
2. 接尾辞木 $ST(T\$)$ 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T_1'とする．
3. 木$T_i' = ST_{Li}(T\$)$ 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 T_i' の葉の総数とし，C_v を v の子の数とする．
4. もし，$L_i + C_v - 1 \leqq 2^l$ を満たすならば，v のすべての子を新たな葉として T_i' に加え，それを新たな文節木候補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T_i'が伸ばせなくなるまで繰り返す．

実験結果（圧縮率比較）

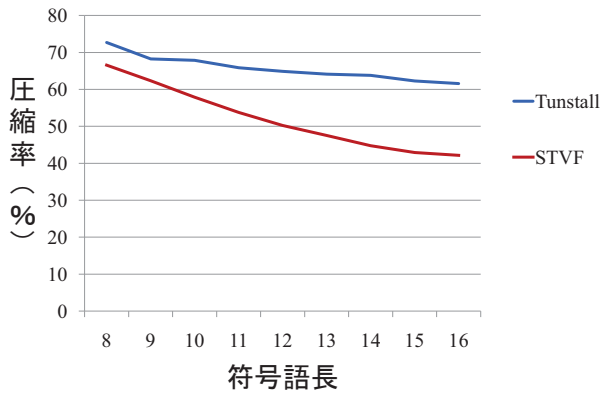
圧縮法	E.coli	bible.txt	world192.txt	dazai.utf.txt	1000000.txt
Huffman符号	25.00%	54.82%	63.03%	57.50%	59.61%
Tunstall符号 (8)	27.39%	72.70%	85.95%	100.00%	76.39%
Tunstall符号 (12)	26.47%	64.89%	77.61%	69.47%	68.45%
Tunstall符号 (16)	26.24%	61.55%	70.29%	70.98%	65.25%
ST-VF符号 (8)	25.09%	66.59%	80.76%	73.04%	74.25%
ST-VF符号 (12)	25.10%	50.25%	62.12%	52.99%	68.90%
ST-VF符号 (16)	28.90%	42.13%	49.93%	41.37%	78.99%

（）内は符号長（ビット）

テキスト	サイズ (byte)	|Σ|	内容
E.coli	4638690	4	Complete genome of the E. Coli bacterium
bible.txt	4047392	63	The King James version of the bible
world192.txt	2473400	94	The CIA world fact book
dazai.utf.txt	7268943	141	The all works of Osamu Dazai (UTF-8 encoded)
1000000.txt	1000000	26	The random string automatically generated

※Canterbury Corpus(http://corpus.canterbury.ac.nz/), 日本文学電子図書館(http://www.j-texts.com/) より

実験結果（圧縮率）



北海道大学 Hokkaido University

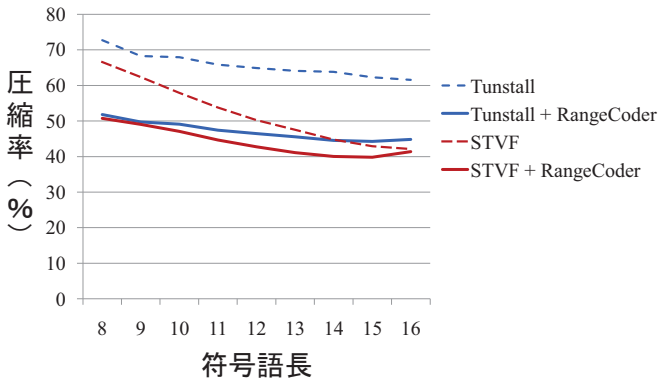
Range coderとの組み合わせによる圧縮率向上

- 圧縮手法
  - Tunstall
  - STVF
  - Tunstall + Range Coder
  - STVF + Range Coder
- データ
  - King James 版聖書 (4MB, |Σ |=63)
- 実験環境
  - CPU: Intel® Xeon® プロセッサ3.00GHz デュアルコア
  - メモリ: 12GB
  - OS: Red Hat Enterprise Linux ES Release 4
- 符号語長
  - 8~16ビット

各手法での圧縮率，圧縮時間，展開時間を比較

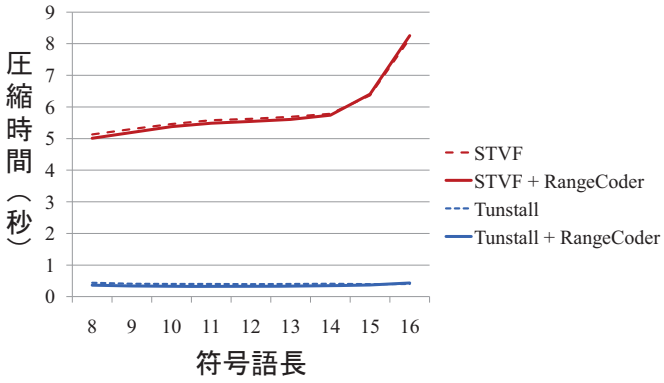
北海道大学 Hokkaido University

実験結果（符号長-圧縮率）



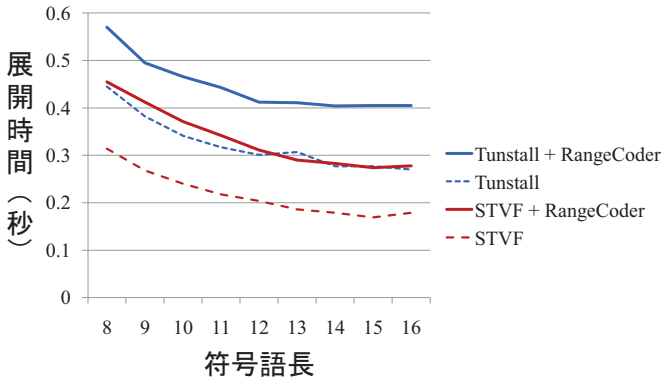
北海道大学 Hokkaido University

実験結果（符号長-圧縮時間）



北海道大学 Hokkaido University

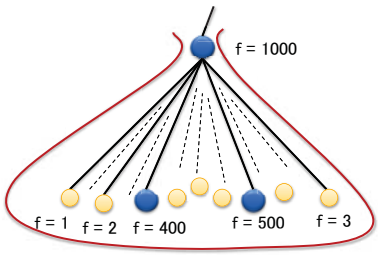
実験結果（符号長-展開時間）



北海道大学 Hokkaido University

オリジナルSTVF符号の問題点

- 出現頻度の高いノードを選択しても
  その子すべての頻度が高いとは限らない



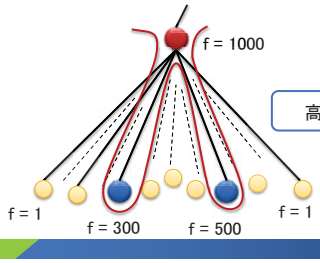
すべての子を無差別に分節木に加えてしまう

北海道大学 Hokkaido University

Almost instantaneous符号化による改善

AIVF符号（Yamamoto&Yokoo 2001）

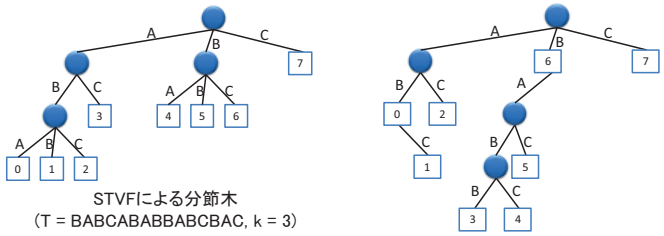
- 出現頻度順に1ノードずつ分節木に追加
  - 内部ノードへの符号割り当てを許すことで実現
  - 符号化の仕方は，1文字先読みで行われる
    - 分節木をたどれなくなったら符号語を出力
    - 根から探索を再開するときは，たどれなくなった文字から始める



高頻度なものだけを選択

オリジナルと改善手法の分節木の違い

- 改善手法では，頻度の低く短い葉ノードが取り除かれ，より長い文字列に符号語が割り当てられやすくなる



STVFによる分節木
(T = BABCABABBABCBAC, k = 3)

改善手法による分節木

より長い文字列が分節木に含まれやすい

計算機実験による評価

- データ
  - Bible.txt (The Canterbury Corpus), 3.85MB
- 方法
  - STVFと提案手法を比較，符号語: 16 bit
- 結果
  - 圧縮率で18.7%，照合速度で22.2%改善

手法	圧縮時間	圧縮率	照合速度
STVF	6109ms	42.1%	7.27ms
改善手法	6593ms	34.2%	5.67ms

環境: Intel Core 2 Duo T7700, 2GB RAM, Windows Vista, Visual C++ 2008

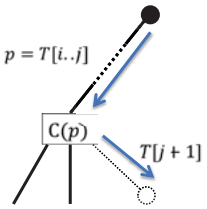
分節木の繰り返し学習による改善

- テキスト中に高い頻度で現れていても，ブロックとして高い頻度で現れるとは限らない！？
- テキストを「最適に」分割する分節木は作れるか？
  - ⇒ 分節木に登録する文字列を，圧縮される際に実際に使われる頻度を基準に選ぶ
  - ⇒ 選んだ単語の集合によって，テキストの分割が変化する
  - ⇒ 卵が先か鶏が先か！ ⇒ NP完全以上

- できるだけ良い分節木を何とか作れないだろうか？
  - テキストを繰り返し圧縮してみて，分節木をブラッシュアップしてみよう！

ブラッシュアップのアイデア

- VF符号化してみて，実際にはあまり使われなかったノードを入れ替えることで分節木を改善しよう！
- 評価方法:
  - ノード u の貢献度を実際に符号化してみて計算 [accept count]
  - 分節木にないノード v の期待度を推定 [failure count]
  - 「uの貢献度 < vの期待度」ならばそれらを入れ替える



$A(p) \leftarrow A(p) + 1$

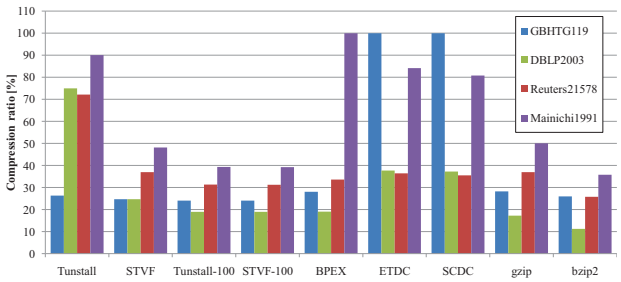
$F(p \cdot T[j+1]) \leftarrow F(p \cdot T[j+1]) + 1$

実験

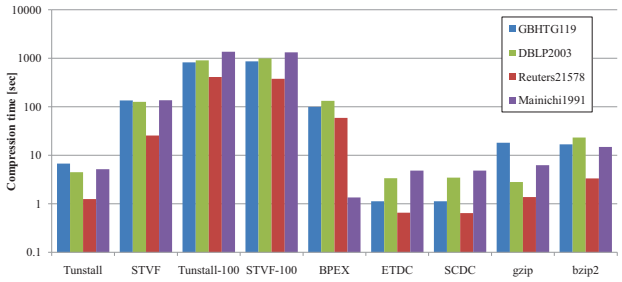
Texts	Size(byte)	|Σ|	Contents
GBHTG119	81,173,787	4	DNA sequences
DBLP2003	90,510,236	97	XML data
Reuters-21578	18,805,335	103	English texts
Mainichi1991	78,911,178	256	Japanese texts(UTF-8)

- 各テキストについて，BPEX, ETDC, SCDC, gzip bzip2との圧縮性能の比較を行った
- C++ compiled by g++ of GNU v 3.4
- Intel Xeon® 3GHz and 12GB of RAM, running Red Hat Enterprise Linux ES Release 4.

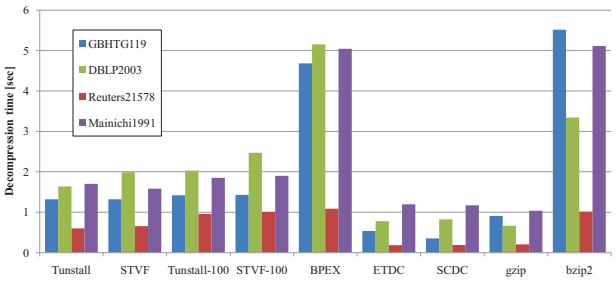
圧縮率



圧縮速度



展開速度



まとめ

- VF符号にこだわって，圧縮率の改善に取り組んだ
  - 接尾辞木を刈り込んだ木を分節木として用いるSTVF符号を提案
  - Almost instantaneousによる改善
  - 分節木の繰り返し学習による改善
- VF符号でありながら，GzipやBPEXなみの圧縮率を達成した
- 今後の課題
  - Bzip2に追いつく！
  - 改善STVF符号上での圧縮パターン照合
    - BM法に基づくアルゴリズムの実装