lex-parse の圧縮感度

中島 祐人　クップル ドミニク　舩越 満　稲永 俊介

概 要

任意の文字列 X について，X を 1 文字編集して得られる任意の文字列を Y と表す．文字列のある圧縮法 C について，C による X の圧縮表現のサイズと Y の圧縮表現のサイズの最悪時ギャップを，圧縮法 C の圧縮感度と呼ぶ．圧縮感度は，文字列データの微小な変化（動的な編集，ノイズ）に対する圧縮法の頑健性を表す指標の一つである．本研究では，lex-parse と呼ばれる圧縮法に対して，圧縮感度のタイトな上下界を与える．

キーワード： 文字列圧縮，lex-parse，フィボナッチ語

1　はじめに

1.1　lex-parse と圧縮感度

辞書式圧縮 (dictionary compression) は可逆圧縮の一種であり，反復性の高い文字列に対して有効な圧縮の枠組みである．例えば，文字列の前方へのポインタで表現する LZ77 は gzip と呼ばれる圧縮ツールの核をなし，辞書式圧縮法の代表格である．また，macro scheme [5, 6] と呼ばれる辞書式圧縮は，前方に限らない任意の位置の出現へのポインタで文字列を表現しており，LZ77 の一般化である．一般に macro scheme は一意に定まらず，最小サイズの macro scheme を求める問題は NP 困難であることが知られている．lex-parse [4] は macro scheme の一つを与える文字列分解であり，線形時間で計算可能である．任意の文字列 w に対して，最小の macro scheme のサイズを $b(w)$，lex-parse のサイズを $v(w)$ としたとき，$v(w) \in O(b(w)\log(n/b(w)))$ が成り立つことが知られている．

ある辞書式圧縮法 C について，C を用いて文字列 w を圧縮したときのサイズを $C(w)$ で表すこととする．圧縮法 C の圧縮感度とは，以下のように定義される指標である．

$$\mathsf{MS}(C,n) = \max_{w_1 \in \Sigma^n} \{C(w_2)/C(w_1) \mid w_2 \in \mathsf{ed}(w_1)\}.$$

ただし，$\mathsf{ed}(w)$ は，文字列 w と編集距離が 1 である文字列の集合である．圧縮感度は，文字列データの微小な変化（動的な編集，ノイズ）に対する圧縮法の頑健性を表す指標の一つとして赤木ら [1] によって提案され，様々な圧縮

法や圧縮スキームに対して，感度の上下界が明らかにされている．本稿では，比による感度（乗算感度）のみを扱うが，差分による感度（加算感度）も同様に提案され [1]，多くの圧縮法に対して上下界が明らかにされている．

1.2 主結果

本研究では，lex-parse の圧縮感度の上下界を与える．

定理 1 (乗算感度の上界)**.** 長さ n の任意の文字列 w_1 に対して，以下が成り立つ．

$$\max\{v(w_2)/v(w_1) \mid w_2 \in \mathsf{ed}(w_1)\} \in O(\log(n/b(w_1))).$$

定理 2 (乗算感度の下界)**.** w を長さ n の文字列とする．$v(w) \in O(1)$ かつ $v(w') \in \Theta(\log n)$ を満たす文字列 w, w' の系列が存在する（ただし，w' は $w' \in \mathsf{ed}(w)$ を満たす文字列である）．

2 準備

2.1 文字列

Σ でアルファベットを表し，Σ^* の要素を文字列と呼ぶ．文字列 w の長さを $|w|$ と表す．また，長さ 0 の文字列 ε を空文字列と呼ぶ．文字列 $w = xyz$ について，x, y, z をそれぞれ w の接頭辞，部分文字列，接尾辞と呼ぶ．文字列 w の i 番目の文字を $w[i]$ で表す $(1 \le i \le |w|)$．任意の文字列 w と二つの整数 i, j $(1 \le i \le j \le |w|)$ について，$w[i..j]$ は位置 i を開始位置とし，位置 j を終了位置とする部分文字列を表すとする．言い換えると，$w[i..j] = w[i] \cdots w[j]$ である．便宜上，$i > j$ であるとき $w[i..j] = \varepsilon$ とする．任意の異なる二つの文字 $a, b \in \Sigma$ について，a が b より小さいことを $a \prec b$ で表す．

2.2 lex-parse と macro scheme

lex-parse は，接尾辞配列 (suffix array)，逆接尾辞配列 (inverse suffix array)，最長共通接頭辞配列 (longest common prefix array) によって定義される文字列分解である [4].

定義 1. 長さ n の文字列 w の非空なすべての接尾辞を辞書順に整列したときの接尾辞の開始位置の列からなる配列 SA を w の接尾辞配列と呼ぶ．また $SA[i] = j$ に対して，$ISA[j] = i$ を満たす配列 ISA を w の逆接尾辞配列と呼ぶ．さらに，$LCP[i]$ が $w[SA[i-1]..n]$ と $w[SA[i]..n]$ の最長共通接頭辞の長さを格納する配列 LCP を w の最長共通接頭辞配列と呼ぶ（ただし，$LCP[1] = 0$ とする）．

lex-parse	1	2	3	4	5	6	7	8	9
	a	a	b	a	b	a	a	b	a

SA	ISA	LCP	整列された接尾辞列
9	3	0	a
6	6	1	a a b a
1	9	4	a a b a b a a b a
7	5	1	a b a
4	8	3	a b a a b a
2	2	3	a b a b a a b a
8	4	0	b a
5	7	2	b a a b a
3	1	2	b a b a a b a

図 1: 文字列 aababaaba の SA，ISA，LCP，lex-parse を示す.

定義 2. 文字列 w の分解 $W_1, \ldots, W_v$ が w の lex-parse であるとは，任意の i について $W_i = w[s_i..s_i + |W_i| - 1]$ が，$|W_i| = LCP[ISA[s_i]]$ を満たす，または，$|W_i| = 1$ かつ $LCP[ISA[s_i]] = 0$ を満たすことである（図 1 参照）. また，v を lex-parse のサイズと呼ぶ.

macro scheme [5, 6] は，以下のように定義される文字列のコンパクトな表現の一つである.

定義 3. 文字列 w の macro scheme は，以下の二種類の規則による w の表現である.

1. $w[i..j] \leftarrow w[i'..j']$
2. $w[i] \leftarrow a\ (\in \Sigma)$

規則 (1) は，部分文字列 $w[i..j]$ は $[i'..j']$ に出現を持ち，コピーされることを意味する．規則 (2) は，$w[i] = a$ であることを意味する．ここでは，元の文字列を一意に復元可能な macro scheme のみを考えることとする．また，規則の数を，macro scheme のサイズと呼ぶ．最小サイズの macro scheme を求める問題は NP 困難であることが知られている．lex-parse を用いることで，macro scheme の一つを得ることができる．lex-parse の定義の後者の条件による部分文字列は規則 (2)，前者の条件による部分文字列は規則 (1) を

用いれば良い．図 1 の例では，

$$
\begin{aligned}
w[9] &\leftarrow a \\
w[8] &\leftarrow b \\
w[1..4] &\leftarrow w[6..9] \\
w[5..6] &\leftarrow w[8..9] \\
w[7..7] &\leftarrow w[1..1]
\end{aligned}
$$

のように lex-parse サイズの macro scheme を構成できる．

3 lex-parse の圧縮感度の上下界

本章では，1.2 節で述べた主結果の概要を紹介する．

3.1 下界

下界を与えるために，フィボナッチ語と呼ばれる文字列の系列を用いる．

定義 4 (フィボナッチ語 (cf. [3]))．アルファベット $\{a, b\}$ 上の k 番目のフィボナッチ語 F_k は以下のように定義される．

- $F_1 = b, F_2 = a$
- $F_k = F_{k-1}F_{k-2}$ $(k \geq 3)$

以下に，8 番目までのフィボナッチ語を示す．

$$
\begin{aligned}
F_1 &= b \\
F_2 &= a \\
F_3 &= ab \\
F_4 &= aba \\
F_5 &= abaab \\
F_6 &= abaababa \\
F_7 &= abaababaabaab \\
F_8 &= abaababaabaababaababa
\end{aligned}
$$

任意の $k \geq 3$ について，F_{2k} から各編集操作により得られる文字列を以下の通り与える（ただし，\$ は $\$ \prec a \prec b$ を満たす文字とする）．

定義 5 (下界を与える文字列系列)．FS_{2k}, FD_{2k}, FI_{2k} はそれぞれ，F_{2k} から置換，削除，挿入操作により得られる文字列である．

置換　$FS_{2k} = F_{2k}[1..|F_{2k}| - 2] \cdot aa$

削除　$FD_{2k} = F_{2k}[1..|F_{2k}| - 2] \cdot a$

挿入　$FI_{2k} = F_{2k}[1..|F_{2k}| - 2] \cdot \$ba$

直感的には，FS_{2k} は最後の b を a に置換，FD_{2k} は最後の b を削除，FI_{2k} は最後の b の直前に \$ に挿入して得られる文字列である．偶数番目のフィボナッチ語 F_{2k} について，lex-parse のサイズは定数であることが知られている [4]．したがって，上記の各文字列の lex-parse サイズが $\Theta(\log n)$ $(= \Theta(k))$ であることを示すことで，定理 2 が得られる．

補題 1. 任意の $k \geq 3$ について，$v(FS_{2k}) = 2k - 2$.

補題 2. 任意の $k \geq 3$ について，$v(FD_{2k}) = 2k - 2$.

補題 3. 任意の $k \geq 3$ について，$v(FI_{2k}) = 2k$.

各補題の証明には，フィボナッチ語の性質および Lyndon 分解 [2] の性質を用いる．Lyndon 文字列は，文字列の辞書式順序に依存した文字列分解の一種であり，任意の文字列に対して一意に定まる．各文字列に共通する接頭辞 $F_{2k}[1..|F_{2k}| - 2]$ の Lyndon 分解の構造を利用することで，各文字列の lex-parse の構造を明らかにできる．本稿では証明の詳細は割愛する．

3.2　上界

lex-parse と macro scheme について，以下の関係が成り立つことが知られている．

補題 4 ([4])**.** 任意の文字列 w について，

$$v(w) \in O(b(w) \log(n/b(w)))$$

が成り立つ．

補題 5 ([1])**.** $w_2 \in \mathsf{ed}(w_1)$ を満たす任意の文字列 w_1, w_2 について，

$$b(w_1) \leq 2b(w_2)$$

が成り立つ．

これらの補題を用いると，定理 1 が得られる．

$$\begin{aligned}\frac{v(w_2)}{v(w_1)} \leq \frac{v(w_2)}{b(w_1)} &\in O\left(\frac{b(w_2)\log(n/b(w_2))}{b(w_1)}\right) \\ &\subseteq O\left(\frac{b(w_1)\log(n/b(w_1))}{b(w_1)}\right) \\ &= O(\log(n/b(w_1))).\end{aligned}$$

フィボナッチ語の macro scheme の最小サイズもまた定数であるため，この上界は先の下界に対してタイトである．

4 おわりに

本研究では，編集操作による lex-parse サイズ比についてタイトな上下界を与えた．様々な圧縮法や圧縮スキームについて，感度の上下界が明らかにされているが，超定数でタイトな上下界が知られている圧縮法は非常に例が少なく，その点で興味深い新たな知見であると言える．

参考文献

[1] T. Akagi, M. Funakoshi, and S. Inenaga. Sensitivity of string compressors and repetitiveness measures. *Information and Computation*, 291:104999, 2023.

[2] K. T. Chen, R. H. Fox, and R. C. Lyndon. Free differential calculus. iv. the quotient groups of the lower central series. *Annals of Mathematics*, 68(1):81–95, 1958.

[3] M. Lothaire. *Applied combinatorics on words*, volume 105. Cambridge University Press, 2005.

[4] G. Navarro, C. Ochoa, and N. Prezza. On the approximation ratio of ordered parsings. *IEEE Trans. Inf. Theory*, 67(2):1008–1026, 2021.

[5] J. A. Storer and T. G. Szymanski. The macro model for data compression (extended abstract). In R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, and A. V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 30–39. ACM, 1978.

[6] J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *J. ACM*, 29(4):928–951, 1982.