Vol.2023-AL-191 No.4
2023/1/20

# アルファベット順による lex-parse サイズ比

中島 祐人[1,a)]　クップル ドミニク[2,b)]　舩越 満[1,c)]　稲永 俊介[1,d)]

**概要**：文字列の圧縮表現の一つに macro scheme と呼ばれる表現方法がある．macro scheme では，部分文字列を別の位置での出現へのポインタで表現することにより，文字列をコンパクトに表現する．lex-parse は，macro scheme の一つを与える文字列分解であり，辞書順依存の構造である．つまり，辞書順の変化により lex-parse はその形を変えうる．本研究では，フィボナッチ語と呼ばれる規則的な文字列の系列の lex-parse について考え，異なる辞書順による lex-parse の変化およびそのサイズ比について議論する．

**キーワード**：文字列圧縮，lex-parse，フィボナッチ語

## 1. はじめに

本研究では，辞書式順序に基づいた文字列分解の一つである lex-parse について，アルファベット順による（圧縮）サイズ比について議論する．

### 1.1 アルファベット順による lex-parse 最小化

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 [1], [2] 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 [3] 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．

lex-parse は辞書式順序を用いて定義される分解であり，異なる順序に対する分解は一般に異なる．つまり，同じ文字列に対して lex-parse のサイズが小さくなる順序を選ぶことができれば，より効率的に圧縮できることを意味する．ここで，アルファベット $\Sigma$ 上の全順序 $\prec$ における文字列 $w$ の lex-parse のサイズを $v(w, \prec)$ とすると，先の問題は，文字列 $w \in \Sigma^*$ に対して，$v(w, \prec)$ を最小にする $\Sigma$ 上の全順序 $\prec$ を求める問題（lex-parse 最小化問題）として定式化される．

### 1.2 主結果

本研究では，lex-parse 最小化問題の効率的解法を開発するアプローチとして，異なるアルファベット順における $v(w, \prec)$ の比の最大値の上下界を与える．$\Sigma$ 上の全順序集合 $A$ について，

$$\mathsf{LP}(w) = \max_{\prec_1, \prec_2 \in A} \frac{v(w, \prec_1)}{v(w, \prec_2)}$$

と定義すると，主結果は以下の通りである．

**定理 1.** 長さ $n$ の任意の文字列 $w$ について，$\mathsf{LP}(w) \in \Theta(\log(n/b(w)))$ が成り立つ．

特に，下界についてはフィボナッチ語と呼ばれる文字列の系列を用いることで示すことができる．

### 1.3 関連研究

アルファベット順の変更により文字列構造を最適化する問題に対する研究は，Burrows-Wheeler（BW）変換や Lyndon 分解に対して以下のことがこれまでに明らかにされている．BW 変換は lex-parse と同様に，接尾辞の辞書順の関係により定まる文字列の変換である．変換後の文字列は同種の文字が連続して現れやすく，連長圧縮によってコンパクトに表現できると期待される．BW 変換は，索引構造をはじめとした様々な文字列処理に利用されるばかり

[1] 九州大学 大学院システム情報科学研究院
[2] 東京医科歯科大学 M&D データ科学センター
[a)] nakashima.yuto.003@m.kyushu-u.ac.jp
[b)] koeppl.dsc@tmd.ac.jp
[c)] mitsuru.funakoshi@inf.kyushu-u.ac.jp
[d)] inenaga@inf.kyushu-u.ac.jp

でなく，実用面においても bzip2 と呼ばれる圧縮ツールに用いられるなど，重要な技術である．この BW 変換の連長圧縮サイズの最小化問題について，決定問題が NP 完全であり，さらに最小化問題は APX 困難であることが知られている [4]．また，Lyndon 分解と呼ばれる辞書式順序に基づいた文字列分解について，分解サイズの最小/最大化問題を Clare と Daykin は提案し，発見的解法を与えている [5]．その後も同問題に対するいくつかの発見的解法が提案されている [6], [7]．一方で Gibney と Thankachan は，Lyndon 分解の最小/最大化問題について，決定問題が NP 完全であることや，ユニークゲーム予想 (Unique Games Conjecture) の下での近似困難性について述べている [8]．

## 2. 準備

### 2.1 文字列

$\Sigma$ でアルファベットを表し，$\Sigma^*$ 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 $\varepsilon$ 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 $(1 \leq i \leq |w|)$．任意の文字列 $w$ と二つの整数 $i, j$ $(1 \leq i \leq j \leq |w|)$ 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 [3]．

**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）．

**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 [1], [2] 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．

| 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．

(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，

$$w[9] \leftarrow a$$
$$w[8] \leftarrow b$$
$$w[1..4] \leftarrow w[6..9]$$
$$w[5..6] \leftarrow w[8..9]$$
$$w[7..7] \leftarrow w[1..1]$$

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

## 3. lex-parse サイズ比の上下界

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.** 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．

- $F_0 = b, F_1 = a$
- $F_k = F_{k-1}F_{k-2}$ $(k \geq 2)$

以下に，7 番目までのフィボナッチ語を示す．

$$F_0 = b$$
$$F_1 = a$$
$$F_2 = ab$$
$$F_3 = aba$$
$$F_4 = abaab$$
$$F_5 = abaababa$$
$$F_6 = abaababaabaab$$
$$F_7 = abaababaabaababaabaaba$$

次に示す補題は，フィボナッチ語の lex-parse サイズについて述べている．また，フィボナッチ語 $F_k$ の macro scheme のサイズは定数であることが知られているため [3], [9]，下界 $\mathsf{LP}(w) \in \Omega(\log(n/b(w)))$ を得ることができる．

**補題 1.** アルファベット $\{a, b\}$ 上の $k$ 番目のフィボナッチ語 $F_k$ について，$F_k$ の lex-parse サイズ $v(F_k)$ は，
(1) $k$ が奇数かつ $a \prec b$ のとき $v(F_k) = \lceil \frac{k}{2} \rceil + 1$,
(2) $k$ が奇数かつ $a \succ b$ のとき $v(F_k) = 4$,
(3) $k$ が偶数かつ $a \prec b$ のとき $v(F_k) = 4$,
(4) $k$ が偶数かつ $a \succ b$ のとき $v(F_k) = \frac{k}{2} + 1$.

この補題の (3) は，Köppl と I のフィボナッチ語の接尾辞配列に関する定理 [10] より導くことができる．(2) についても同定理に対してフィボナッチ語の偶奇性による性質を用いることで導くことができる．(1) は Navarro らによって示されているが [3]，フィボナッチ語の巡回に関する性質を用いているため，この結果から (4) を直ちに導くことは容易ではないと考えられる．そこで本研究では (1) に対して，巡回を用いた議論なしの別証明を与えた．この証明およびフィボナッチ語の偶奇性による性質から (4) の結果も導くことができる．本稿では証明の詳細は割愛する．

### 3.2 上界

lex-parse サイズの macro scheme が存在するため，$v(w) \geq b(w)$ が成り立つことは明らかである．また，$v(w), b(w)$ に関する以下の関係より，lex-parse サイズ比の上界 $\mathsf{LP}(w) \in O(\log(n/b(w)))$ を導くことができる．

**補題 2** ([3])**.** 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．

つまり，以下を満たすある定数 $c$ が存在する．

$$\frac{v(w, \prec_1)}{v(w, \prec_2)} \leq \frac{c \cdot b(w) \log(n/b(w))}{b(w)} = c \cdot \log(n/b(w))$$

## 4. おわりに

本研究では，アルファベット順による lex-parse サイズ比についてタイトな上下界を与えた．このような枠組みの問題は，辞書式順序依存である様々な文字列構造に対して考えることができるが，著者らの知る限り，lex-parse はタイトな上下界を持つことが明らかになった初めての文字列構造である．

### 参考文献

[1] Storer, J. A. and Szymanski, T. G.: The Macro Model for Data Compression (Extended Abstract), *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA* (Lipton, R. J., Burkhard, W. A., Savitch, W. J., Friedman, E. P. and Aho, A. V., eds.), ACM, pp. 30–39 (online), DOI: 10.1145/800133.804329 (1978).

[2] Storer, J. A. and Szymanski, T. G.: Data compression via textual substitution, *J. ACM*, Vol. 29, No. 4, pp. 928–951 (online), DOI: 10.1145/322344.322346 (1982).

[3] Navarro, G., Ochoa, C. and Prezza, N.: On the Approximation Ratio of Ordered Parsings, *IEEE Trans. Inf. Theory*, Vol. 67, No. 2, pp. 1008–1026 (online), DOI: 10.1109/TIT.2020.3042746 (2021).

[4] Bentley, J. W., Gibney, D. and Thankachan, S. V.: On the Complexity of BWT-Runs Minimization via Alphabet Reordering, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)* (Grandoni, F., Herman, G. and Sanders, P., eds.), LIPIcs, Vol. 173, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 15:1–15:13 (online), DOI: 10.4230/LIPIcs.ESA.2020.15 (2020).

[5] Clare, A. and Daykin, J. W.: Enhanced string factoring from alphabet orderings, *Inf. Process. Lett.*, Vol. 143, pp. 4–7 (online), DOI: 10.1016/j.ipl.2018.10.011 (2019).

[6] Clare, A., Daykin, J. W., Mills, T. and Zarges, C.: Evolutionary search techniques for the Lyndon factorization of biosequences, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019* (López-Ibáñez, M., Auger, A. and Stützle, T., eds.), ACM, pp. 1543–1550 (online), DOI: 10.1145/3319619.3326872 (2019).

[7] Major, L., Clare, A., Daykin, J. W., Mora, B., Gamboa, L. J. P. and Zarges, C.: Evaluation of a Permutation-Based Evolutionary Framework for Lyndon Factorizations, *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I* (Bäck, T., Preuss, M., Deutz, A. H., Wang, H., Doerr, C., Emmerich, M. T. M. and Trautmann, H., eds.), Lecture Notes in Computer Science, Vol. 12269, Springer, pp. 390–403 (online), DOI: 10.1007/978-3-030-58112-1_27 (2020).

[8] Gibney, D. and Thankachan, S. V.: Finding an Optimal Alphabet Ordering for Lyndon Factorization Is Hard, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)* (Bläser, M. and Monmege, B., eds.), LIPIcs, Vol. 187, Schloss Dagstuhl - Leibniz-

Zentrum für Informatik, pp. 35:1–35:15 (online), DOI: 10.4230/LIPIcs.STACS.2021.35 (2021).

[9] Mantaci, S., Restivo, A. and Sciortino, M.: Burrows-Wheeler transform and Sturmian words, *Inf. Process. Lett.*, Vol. 86, No. 5, pp. 241–246 (online), DOI: 10.1016/S0020-0190(02)00512-4 (2003).

[10] Köppl, D. and I, T.: Arithmetics on Suffix Arrays of Fibonacci Words, *Combinatorics on Words - 10th International Conference, WORDS 2015, Kiel, Germany, September 14-17, 2015, Proceedings* (Manea, F. and Nowotka, D., eds.), Lecture Notes in Computer Science, Vol. 9304, Springer, pp. 135–146 (online), DOI: 10.1007/978-3-319-23660-5_12 (2015).