Vol.2020-MPS-127 No.3
2020/3/2

Lyndon文字列を用いた文法圧縮に基づく自己索引

鶴田 和弥[1,a)]　クップル ドミニク[1,2,b)]　中島 祐人[1,c)]　稲永 俊介[1,3,d)]　坂内 英夫[1,e)]
竹田 正幸[1,f)]

概要：Lyndon 木 (Barcelo, 1990) より発想を得た *Lyndon SLP* という SLP の新しいクラスを提案する．長さ m のパターン P に対して $locate(P)$ を $\mathcal{O}(m \lg \sigma/w + \lg m \lg n + \mathrm{occ} \lg g)$ 時間でサポートする $\mathcal{O}(g)$ ワード領域の自己索引を提案する．ここで n は入力文字列 T の長さ, g は T の Lyndon SLP のサイズ, σ アルファベットサイズ, occ は出力の数である．自己索引は T から $\mathcal{O}(n + g \lg g)$ 時間で構築が可能である．

キーワード：文法圧縮, 自己索引

Grammar-compressed Self-index with Lyndon Words

Abstract: We introduce a new class of straight-line programs (SLPs), named the *Lyndon SLP*, inspired by the Lyndon trees (Barcelo, 1990). Based on this SLP, we propose a self-index data structure of $\mathcal{O}(g)$ words of space that can be built from a string T in $\mathcal{O}(n + g \lg g)$ time, supporting $locate(P)$ in $\mathcal{O}(m \lg \sigma/w + \lg m \lg n + \mathrm{occ} \lg g)$ time for a pattern P of length m, where n is the length of T, g is the size of the Lyndon SLP for T, σ is the alphabet size, w is the computer word length, and occ is the number of occurrences of P in T.

Keywords: Grammar-based Compression, Self-index

1. Introduction

A context-free grammar is said to *represent* a string T if it generates the language consisting of T and only T. Grammar-based compression [17] is, given a string T, to find a small size description of T based on a context-free grammar that represents T. The grammar-based compression scheme is known to be most suitable for compressing *highly-repetitive strings*. Due to its ease of manipulation, grammar-based representation of strings is a frequently used model for *compressed string processing*, where the aim is to efficiently process compressed strings without explicit decompression. Such an approach allows for theoretical and practical speed-ups compared to a naive decompress-then-process approach.

A *self-index* is a data structure that is a full-text index, i.e., supports various pattern matching queries on the text, and also provides random access to the text, usually without explicitly holding the text itself. Examples are the compressed suffix array [13], [14], [19], the compressed compact suffix array [23], and the FM index [12].[*1] These self-indices are, however, unable to fully exploit the redundancy of highly repetitive strings. To exploit such repetitiveness, Claude and Navarro [8] proposed the first self-index based on grammar-based compression. The method is based on a *straight-line program (SLP)*, a context-free grammar representing a single string in the Chomsky normal form. Several grammar-based self-indexes have been

[1] 九州大学
Kyushu University
[2] 日本学術振興会
Japan Society for Promotion of Science
[3] 国立研究開発法人科学技術振興機構 さきがけ
PRESTO, Japan Science and Technology Agency
[a)] kazuya.tsuruta@inf.kyushu-u.ac.jp
[b)] dominik.koeppl@inf.kyushu-u.ac.jp
[c)] yuto.nakashima@inf.kyushu-u.ac.jp
[d)] inenaga@inf.kyushu-u.ac.jp
[e)] bannai@inf.kyushu-u.ac.jp
[f)] takeda@inf.kyushu-u.ac.jp

[*1] Navarro and Mäkinen [28] published an excellent survey on this topic.

proposed [9], [30], [37], [38].

In this paper, we first introduce a new class of SLPs, named the *Lyndon SLP*, inspired by the Lyndon tree [4]. We then propose a self-index structure of $\mathcal{O}(g)$ words of space that can be built from a string T in $\mathcal{O}(n + g \log g)$ time. The proposed self-index supports *locate*(P) in $\mathcal{O}(m \lg \sigma/w + \lg m \lg n + \mathrm{occ} \lg g)$ time for a pattern P of length m, where n is the length of T, g is the size of the Lyndon SLP for T, σ is the alphabet size, w is the computer word length and occ is the number of occurrences of P in T.

1.1 Related work

The *smallest grammar problem* is, given a string T, to find the context-free grammar G representing T with the smallest possible size, where the *size* of G is the total length of the right-hand sides of the production rules in G. Since the smallest grammar problem is NP-hard [36], many attempts have been made to develop small-sized context-free grammars representing a given string T. LZ78 [40], LZW [39], Sequitur [29], Sequential [17], LongestMatch [17], Re-Pair [20], and Bisection [16] are grammars based on simple greedy heuristics. Among them Re-Pair is known for achieving high compression ratios in practice.

Approximations for the smallest grammar have also been proposed. The AVL grammars [31] and the α-balanced grammars [6] can be computed in linear time and achieve the currently best approximation ratio of $O(\lg(|T|/g_T^*))$ by using the LZ77 factorization and the balanced binary grammars, where g_T^* denotes the smallest grammar size for T. Other grammars with linear-time algorithms achieving the approximation $O(\lg(|T|/g_T^*))$ are LevelwiseRePair [33] and Recompression [15]. They basically replace di-grams with a new variable in a bottom-up manner similar to Re-Pair, but use different mechanisms to select the di-grams. On the other hand, LCA [34] and its variants [25], [26], [35] are known as scalable practical approximation algorithms. The core idea of LCA is the *edit-sensitive parsing (ESP)* [10], a parsing algorithm developed for approximately computing the edit distance with moves. The *locally-consistent-parsing (LCP)* [32] is a generalization of ESP. *signature encoding (SE)* [27], developed for equality testing on a dynamic set of strings, is based on LCP and can be used as a grammar-transform method. The ESP index [37], [38] and the SE index [30] are grammar-based self-indexes based on ESP and SE, respectively.

2. Preliminaries

2.1 Notation

Let Σ be an ordered finite *alphabet*. An element of Σ^* is called a *string*. The length of a string S is denoted by $|S|$. The empty string ε is the string of length 0. For a string $S = xyz$, x, y and z are called a *prefix*, *substring*, and *suffix* of S, respectively. A prefix (resp. suffix) x of S is called a *proper prefix* (resp. suffix) of S if $x \neq S$. The i-th character of a string S is denoted by $S[i]$, where $i \in [1..|S|]$. For a string S and two integers i and j with $1 \leq i \leq j \leq |S|$, let $S[i..j]$ denote the substring of S that begins at position i and ends at position j. For convenience, let $S[i..j] = \varepsilon$ when $i > j$.

2.2 Lyndon words and Lyndon trees

Let $\preceq$ denote some total order on Σ, as well as the lexicographic order induced on Σ^*. We write as $u \prec v$ to imply $u \preceq v$ and $u \neq v$ for any $u, v \in \Sigma^*$.

Definition 2.1 (Lyndon Word [22])**.** A non-empty string $w \in \Sigma^+$ is said to be a *Lyndon word* with respect to $\prec$ if $w \prec u$ for every non-empty proper suffix u of w.

Definition 2.2 (Standard Factorization [7], [21])**.** The *standard factorization* of a Lyndon word w with $|w| \geq 2$ is an ordered pair (u, v) of strings u, v such that $w = uv$ and v is the longest proper suffix of w that is also a Lyndon word.

Lemma 2.3 ([5], [21])**.** For a Lyndon word w with $|w| > 1$, the standard factorization (u, v) of w always exists, and the strings u and v are Lyndon words.

The Lyndon tree of a Lyndon word w, defined below, is the full binary tree induced by recursively applying the standard factorization on w.

Definition 2.4 (Lyndon Tree [4])**.** The *Lyndon tree* of a Lyndon word w, denoted by *LTree*(w), is an ordered full binary tree defined recursively as follows:

- if $|w| = 1$, then *LTree*(w) consists of a single node labeled by w;
- if $|w| \geq 2$, then the root of *LTree*(w), labeled by w, has the left child *LTree*(u) and the right child *LTree*(v), where (u, v) is the standard factorization of w.

Figure 1 shows an example of a Lyndon tree for the Lyndon word `aababaababb`.

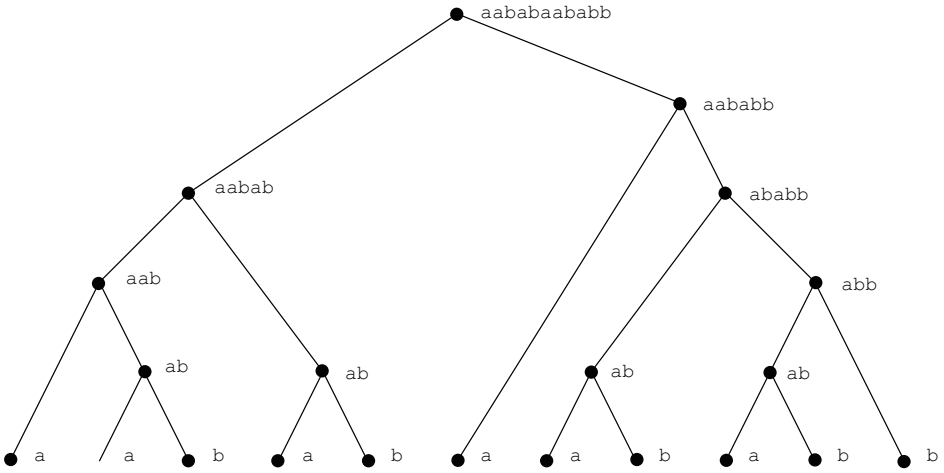


Fig. 1 The Lyndon tree for the Lyndon word `aababaababb` with respect to the order `a` $\prec$ `b`, where each node is accompanied by its label to its right.

2.3 Admissible grammars and straight-line programs (SLPs)

An admissible grammar [17] is a context-free grammar that generates a language consisting only of a single string. Formally, an *admissible grammar (AG)* is a set of production rules $\mathcal{G}_{\mathrm{AG}} = \{X_i \to \mathrm{expr}_i\}_{i=1}^{g}$, where X_i is a *variable* and expr_i is a non-empty string over $\Sigma \cup \{X_1, \ldots, X_{i-1}\}$, called an *expression*. The variable X_g is called the *start symbol*. We denote by $val(X_i)$ the string derived by X_i. We say that an admissible grammar $\mathcal{G}_{\mathrm{AG}}$ *represents* a string T if $T = val(X_g)$. To ease notation, we sometimes associate $val(X_i)$ with X_i. The *size* of $\mathcal{G}_{\mathrm{AG}}$ is the total length of all expressions expr_i.

A *straight-line program* (*SLP*) is an admissible grammar in the Chomsky normal form, namely, each production rule is either of the form $X_i \to a$ for some $a \in \Sigma$ or $X_i \to X_{i_{\mathrm{L}}} X_{i_{\mathrm{R}}}$ with $i > i_{\mathrm{L}}, i_{\mathrm{R}}$. Note that the size of $\mathcal{G}_{\mathrm{SLP}}$ can be as large as $\Theta(2^g)$. This can be seen by the example string $T = \mathtt{a} \cdots \mathtt{a}$ consisting of $n = 2^\ell$ `a`'s, where the smallest SLP $\{X_1 \to \mathtt{a}\} \cup \bigcup_{j=2}^{\ell+1} \{X_j \to X_{j-1}X_{j-1}\}$ has size $2\ell + 1$.

The derivation tree $\mathcal{T}_{\mathcal{G}_{\mathrm{SLP}}}$ of $\mathcal{G}_{\mathrm{SLP}}$ is a labeled ordered binary tree, where each internal node is labeled with a variable in $\{X_1, \ldots, X_g\}$, and each leaf is labeled with a character in Σ. The root node has the start symbol X_g as label. We say that the *height* of $\mathcal{G}_{\mathrm{SLP}}$ is the height of $\mathcal{T}_{\mathcal{G}_{\mathrm{SLP}}}$. An example of the derivation tree of an SLP is shown in Figure 2.

2.4 Grammar irreducibility

An admissible grammar is said to be *irreducible* if it satisfies the following conditions:

C-1. Every variable other than the start symbol is used more than once (**rule utility**);

C-2. All pairs of symbols have at most one non-overlapping occurrence in the right-hand sides of the production rules (**di-gram uniqueness**); and

C-3. Distinct variables derive different strings.

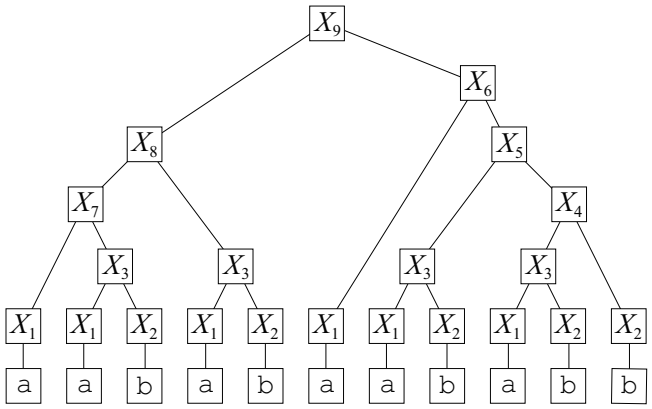
Grammar-based compression is a combination of the *grammar transform* where an admissible grammar G that represents the input string T is computed and the *grammar encoding* where an encoding for G is computed. Kieffer and Yang [17] showed that a combination of an irreducible grammar-transform and a zero order arithmetic code is universal, where a grammar-transform is said to be *irreducible* if the resulting grammars are irreducible.

If an admissible grammar G is not irreducible, we can apply at least one of the following reduction rules [17]:

R-1. Suppose a variable X_i occurs only once in the right-hand sides of the production rules. Then, replace the unique occurrence of X_i with expr_i and remove the rule $X_i \to \mathrm{expr}_i$.

R-2. Suppose there is a string γ of symbols with $|\gamma| \geq 2$ having more than one non-overlapping occurrence in the right-hand sides of the production rules. Then, replace each of the occurrences with variable X_i where $X_i \to \gamma$ is an existing or newly created production rule.

R-3. Suppose there exist two distinct variables X_i, X_j deriving an identical string. Then, replace each occurrence of X_j with X_i in the right-hand sides of the production rules, and remove the production rules



$X_1 \to$ a $\quad X_6 \to X_1X_5$
$X_2 \to$ b $\quad X_7 \to X_1X_3$
$X_3 \to X_1X_2 \quad X_8 \to X_7X_3$
$X_4 \to X_3X_2 \quad X_9 \to X_8X_6$
$X_5 \to X_3X_4$

Fig. 2 *Top*: The derivation tree of the Lyndon SLP $\mathcal{G}_{\mathrm{LYN}}$ representing the Lyndon word $T =$ aababaababb. *Bottom*: The production rules of $\mathcal{G}_{\mathrm{LYN}}$.

containing useless symbols (if these exist).

3. Lyndon SLP

In what follows, we propose a new SLP, called Lyndon SLP. A *Lyndon SLP* is an SLP $\mathcal{G}_{\mathrm{LYN}} = \{X_i \to \mathrm{expr}_i\}_{i=1}^{g}$ representing a Lyndon word, and satisfies the following properties:

- The strings $val(X_i)$ are Lyndon words for all variables X_i.
- The standard factorization of the string $val(X_i)$ is $(val(X_{i_{\mathrm{L}}}), val(X_{i_{\mathrm{R}}}))$ for every rule $X_i \to X_{i_{\mathrm{L}}}X_{i_{\mathrm{R}}}$.
- No pair of distinct variables X_i and X_j satisfies $val(X_i) = val(X_j)$.

The derivation tree (when excluding its leaves) of $\mathcal{T}_{\mathcal{G}_{\mathrm{LYN}}}$ is isomorphic to the Lyndon tree of T (cf. Fig. 2).

The rest of this article is devoted to algorithmic aspects regarding the Lyndon SLP. We study its construction (Sect. 3.1) and propose an index data structure on it (Sect. 4). For that, we work in the word RAM model supporting packing characters of sufficiently small bit widths into a single machine word. Let w denote the machine word size in bits.

We fix a text $T[1..n]$ over an integer alphabet Σ with size $\sigma = n^{\mathcal{O}(1)}$. If T is not a Lyndon word, we prepend T with a character smaller than all other characters appearing in T. We use the text $T :=$ aababaababb as our running example. Let g denote the size $|\mathcal{G}_{\mathrm{LYN}}|$ of the Lyndon SLP $\mathcal{G}_{\mathrm{LYN}}$ of T.

Lemma 3.1 (Algo. 1 of [3])**.** We can construct the Lyndon tree of T in $\mathcal{O}(n)$ time.

3.1 Constructing Lyndon SLPs

The algorithm of Bannai et al. [3] builds the Lyndon tree *online* from right to left. We can modify this algorithm to create the Lyndon SLP of T by storing a dictionary for the rules and a reverse dictionary for looking up rules: Whenever the algorithm creates a new node u, we query the reverse dictionary with u's two children v and w for an existing rule $X \to X_vX_w$, where X_v and X_w are the variables representing v and w. If such a rule exists, we assign u the variable X, otherwise we create a new rule $X_u \to X_vX_w$ and put this new rule into both dictionaries. The dictionaries can be implemented as balanced search trees or hash tables, featuring $\mathcal{O}(n \lg g)$ deterministic construction time or $\mathcal{O}(n)$ expected construction time, respectively.

In the static setting (i.e., we do not work online), deterministic $\mathcal{O}(n)$ time can be achieved by using the enhanced suffix array [1], [24] supporting constant time longest common extension queries. For each node of the Lyndon tree corresponding to $T[i..j]$, associate the pair $(|T[i..j]|, \mathit{rank}(i))$, i.e., its length, as well as the lexicographic rank of the suffix starting at position i. Then, sort all nodes according to this pair. This can be done in $\mathcal{O}(n)$ time using radix sort. By using longest common extension queries between adjacent nodes of equal length in the sorted order, we can determine in $\mathcal{O}(1)$ time per node whether they represent the same string, and if so, assign the same variable or otherwise assign a new variable.

4. Lyndon SLP Based Self-Indices

Given Lyndon SLP of size g, we can build an indexing data structure to report all occurrences of a pattern P of length $m \in (0..n]$ in T. We call this query $\mathit{locate}(P)$. Our data structure is based on the approach of [8] separating the pattern search in the task to locate primary occurrences with an orthogonal range query data structure, and subsequently finding the secondary occurrences with the grammar tree.

Theorem 4.1. Given the Lyndon SLP G of T, there is a data structure using $\mathcal{O}(g)$ words that can be constructed in $\mathcal{O}(n + g \lg g)$ time on it, supporting $\mathit{locate}(P)$ in $\mathcal{O}(m \lg \sigma/w + \lg m \lg n + \mathrm{occ} \lg g)$ time for a pattern P of length m, where n is the length of T, g is the size of G, σ is the alphabet size, w is the computer word length and occ is the number of occurrences of P in T.

5. Conclusion

We introduced a new class of SLPs, named the *Lyndon SLP*, and proposed a self-index structure of $\mathcal{O}(g)$ words of space, which can be built from an input string T in $\mathcal{O}(n + g \lg g)$ time, where n is the length of T and g is the size of the Lyndon SLP for T. By exploiting combinatorial properties on Lyndon SLPs, we showed that $locate(P)$ can be computed in $\mathcal{O}(m \lg \sigma/w + \lg m \lg n + \mathrm{occ} \lg g)$ time for a pattern P of length m, where σ is the alphabet size, w is the computer word length, and occ is the number of occurrences of P. This is better than the $\mathcal{O}((m^2/\epsilon) \lg(\frac{\lg n}{\lg g}) + (m + \mathrm{occ}) \lg g)$ query time of the SLP-index by Claude and Navarro [9] ($0 < \epsilon \leq 1$), which works for a *general* SLP of size g.

We have not implemented the proposed self-index structure, and comparing it with other self-index implementations such as the FM index [11], the LZ index [2], the ESP index [38], or the LZ-end index [18] will be a future work.

参考文献

[1] Abouelhoda, M. I., Kurtz, S. and Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays, *J. Discrete Algorithms*, Vol. 2, No. 1, pp. 53–86 (2004).
[2] Arroyuelo, D. and Navarro, G.: Space-efficient construction of Lempel-Ziv compressed text indexes, *Inf. Comput.*, Vol. 209, No. 7, pp. 1070–1102 (2011).
[3] Bannai, H., I, T., Inenaga, S., Nakashima, Y., Takeda, M. and Tsuruta, K.: The "Runs" Theorem, *SIAM J. Comput.*, Vol. 46, No. 5, pp. 1501–1514 (2017).
[4] Barcelo, H.: On the action of the symmetric group on the Free Lie Algebra and the partition lattice, *Journal of Combinatorial Theory, Series A*, Vol. 55, No. 1, pp. 93–129 (1990).
[5] Bassino, F., Clément, J. and Nicaud, C.: The standard factorization of Lyndon words: an average point of view, *Discrete Mathematics*, Vol. 290, No. 1, pp. 1–25 (2005).
[6] Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A. and abhi shelat: The Smallest Grammar Problem, *IEEE Transactions on Information Theory*, Vol. 51, No. 7, pp. 2554–2576 (2005).
[7] Chen, K. T., Fox, R. H. and Lyndon, R. C.: Free differential calculus, IV. The quotient groups of the lower central series, *Annals of Mathematics*, Vol. 68, No. 1, pp. 81–95 (1958).
[8] Claude, F. and Navarro, G.: Self-Indexed Grammar-Based Compression, *Fundamenta Informaticae*, Vol. 111, No. 3, pp. 313–337 (2011).
[9] Claude, F. and Navarro, G.: Improved Grammar-Based Compressed Indexes, *String Processing and Information Retrieval - 19th International Symposium, SPIRE 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings*, pp. 180–192 (2012).
[10] Cormode, G. and Muthukrishnan, S.: The string edit distance matching problem with moves, *ACM Trans. Algorithms*, Vol. 3, No. 1, pp. 2:1–2:19 (2007).
[11] Ferragina, P., González, R., Navarro, G. and Venturini, R.: Compressed text indexes: From theory to practice, *ACM Journal of Experimental Algorithmics*, Vol. 13, pp. 1.12:1 – 1.12:31 (2008).
[12] Ferragina, P. and Manzini, G.: Opportunistic Data Structures with Applications, *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, IEEE Computer Society, pp. 390–398 (2000).
[13] Grossi, R. and Vitter, J. S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract), *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA* (Yao, F. F. and Luks, E. M., eds.), ACM, pp. 397–406 (2000).
[14] Hon, W., Lam, T. W., Sadakane, K. and Sung, W.: Constructing Compressed Suffix Arrays with Large Alphabets, *Algorithms and Computation, 14th International Symposium, ISAAC 2003, Kyoto, Japan, December 15-17, 2003, Proceedings* (Ibaraki, T., Katoh, N. and Ono, H., eds.), Lecture Notes in Computer Science, Vol. 2906, Springer, pp. 240–249 (2003).
[15] Jez, A.: Approximation of grammar-based compression via recompression, *Theor. Comput. Sci.*, Vol. 592, pp. 115–134 (2015).
[16] Kieffer, J., Yang, E., Nelson, G. and Cosman, P.: Universal Lossless Compression Via Multilevel Pattern Matching, *IEEE Transactions on Information Theory*, Vol. 46, No. 4, pp. 1227–1245 (2000).
[17] Kieffer, J. C. and Yang, E.: Grammar-based codes: A new class of universal lossless source codes, *IEEE Trans. Information Theory*, Vol. 46, No. 3, pp. 737–754 (2000).
[18] Kreft, S. and Navarro, G.: On compressing and indexing repetitive sequences, *Theor. Comput. Sci.*, Vol. 483, pp. 115–133 (2013).
[19] Lam, T. W., Sadakane, K., Sung, W. and Yiu, S.: A Space and Time Efficient Algorithm for Constructing Compressed Suffix Arrays, *Computing and Combinatorics, 8th Annual International Conference, COCOON 2002, Singapore, August 15-17, 2002, Proceedings* (Ibarra, O. H. and Zhang, L., eds.), Lecture Notes in Computer Science, Vol. 2387, Springer, pp. 401–410 (2002).
[20] Larsson, N. J. and Moffat, A.: Offline dictionary-based compression, *Proc. DCC'99*, IEEE Computer Society, pp. 296–305 (1999).
[21] Lothaire, M.: *Combinatorics on Words*, Addison-Wesley (1983).
[22] Lyndon, R. C.: On Burnside's problem, *Transactions of the American Mathematical Society*, Vol. 77, pp. 202–215 (1954).
[23] Mäkinen, V. and Navarro, G.: Compressed Compact Suffix Arrays, *Combinatorial Pattern Matching, 15th Annual Symposium, CPM 2004, Istanbul,Turkey, July 5-7, 2004, Proceedings* (Sahinalp, S. C., Muthukrishnan, S. and Dogrusöz, U., eds.), Lecture Notes in Computer Science, Vol. 3109, Springer, pp. 420–433 (2004).
[24] Manber, U. and Myers, E. W.: Suffix Arrays: A New Method for On-Line String Searches, *SIAM J. Comput.*, Vol. 22, No. 5, pp. 935–948 (1993).
[25] Maruyama, S., Sakamoto, H. and Takeda, M.: An Online Algorithm for Lightweight Grammar-Based Compression, *Algorithms*, Vol. 5, No. 2, pp. 2014–235 (2012).
[26] Maruyama, S., Tabei, Y., Sakamoto, H. and Sadakane, K.: Fully-Online Grammar Compression,

Proc. SPIRE'13, pp. 218–229 (2013).

[27] Mehlhorn, K., Sundar, R. and Uhrig, C.: Maintaining Dynamic Sequences under Equality Tests in Polylogarithmic Time, *Algorithmica*, Vol. 17, No. 2, pp. 183–198 (1997).

[28] Navarro, G. and Mäkinen, V.: Compressed full-text indexes, *ACM Comput. Surv.*, Vol. 39, No. 1, p. 2 (2007).

[29] Nevill-Manning, C. G., Witten, I. H. and Maulsby, D. L.: Compression by Induction of Hierarchical Grammars, *Proc. DCC'94*, pp. 244–253 (1994).

[30] Nishimoto, T., I, T., Inenaga, S., Bannai, H. and Takeda, M.: Dynamic index and LZ factorization in compressed space, *Discrete Applied Mathematics*, Vol. 274 (2019).

[31] Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression, *Theoretical Computer Science*, Vol. 302, No. 1–3, pp. 211–222 (2003).

[32] Sahinalp, S. C. and Vishkin, U.: Data compression using locally consistent parsing, Technical report, UMIACS Technical Report (1995).

[33] Sakamoto, H.: A fully linear-time approximation algorithm for grammar-based compression, *Journal of Discrete Algorithms*, Vol. 3, No. 2–4, pp. 416–430 (2005).

[34] Sakamoto, H., Kida, T. and Shimozono, S.: A Space-Saving Linear-Time Algorithm for Grammar-Based Compression, *Proc. SPIRE'04*, pp. 218–229 (2004).

[35] Sakamoto, H., Maruyama, S., Kida, T. and Shimozono, S.: A Space-Saving Approximation Algorithm for Grammar-Based Compression, *IEICE Transactions*, Vol. 92-D, No. 2, pp. 158–165 (2009).

[36] Storer, J. A.: NP-Completeness Results Concerning Data Compression, Technical Report Technical Report 234, Dept. of Electrical Engineering and Computer Science, Princeton University (1977).

[37] Takabatake, Y., Nakashima, K., Kuboyama, T., Tabei, Y. and Sakamoto, H.: siEDM: An Efficient String Index and Search Algorithm for Edit Distance with Moves, *Algorithms*, Vol. 9, No. 2, p. 26 (2016).

[38] Takabatake, Y., Tabei, Y. and Sakamoto, H.: Improved ESP-index: A Practical Self-index for Highly Repetitive Texts, *Experimental Algorithms - 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29 - July 1, 2014. Proceedings*, pp. 338–350 (2014).

[39] Welch, T. A.: A Technique for High Performance Data Compression, *IEEE Computer*, Vol. 17, pp. 8–19 (1984).

[40] Ziv, J. and Lempel, A.: Compression of Individual Sequences via Variable-length Coding, *IEEE Transactions on Information Theory*, Vol. 24, No. 5, pp. 530–536 (1978).