PAPER *Special Section on Foundations of Computer Science—Frontiers of Theoretical Computer Science—*

# Approximate Frequent Pattern Discovery in Compressed Space*

**Shouhei FUKUNAGA**[†], **Yoshimasa TAKABATAKE**[†], **Tomohiro I**[†], ***Nonmembers***, ***and*** **Hiroshi SAKAMOTO**[†a)], ***Member***

**SUMMARY** A grammar compression is a restricted context-free grammar (CFG) that derives a single string deterministically. The goal of a grammar compression algorithm is to develop a smaller CFG by finding and removing duplicate patterns, which is simply a frequent pattern discovery process. Any frequent pattern can be obtained in linear time; however, a huge working space is required for longer patterns, and the entire string must be preloaded into memory. We propose an *online* algorithm to address this problem approximately within compressed space. For an input sequence of symbols, $a_1, a_2, \ldots$, let $G_i$ be a grammar compression for the string $a_1 a_2 \cdots a_i$. In this study, an online algorithm is considered one that can compute $G_{i+1}$ from $(G_i, a_{i+1})$ without explicitly decompressing $G_i$. Here, let $G$ be a grammar compression for string $S$. We say that variable $X$ approximates a substring $P$ of $S$ within approximation ratio $\delta$ iff for any interval $[i, j]$ with $P = S[i, j]$, the parse tree of $G$ has a node labeled with $X$ that derives $S[\ell, r]$ for a subinterval $[\ell, r]$ of $[i, j]$ satisfying $|[\ell, r]| \geq \delta|[i, j]|$. Then, $G$ solves the frequent pattern discovery problem approximately within $\delta$ iff for any frequent pattern $P$ of $S$, there exists a variable that approximates $P$ within $\delta$. Here, $\delta$ is called the approximation ratio of $G$ for $S$. Previously, the best approximation ratio obtained by a polynomial time algorithm was $\Omega(1/\lg^2|P|)$. The main contribution of this work is to present a new lower bound $\Omega(1/\lg^*|S|\lg|P|)$ that is smaller than the previous bound when $\lg^*|S| < \lg|P|$. Experimental results demonstrate that the proposed algorithm extracts sufficiently long frequent patterns and significantly reduces memory consumption compared to the offline algorithm in the previous work.

***key words:*** *grammar compression, online algorithm, approximate frequent pattern discovery*

## 1. Introduction

### 1.1 Motivation

A *grammar compression* of a string is a context-free grammar (CFG) that derives only the string. In recent decades, various grammar compression algorithms that show good performance, particularly for a *repetitive string* in which multiple long identical patterns (substrings) can be observed, have been proposed. Such data are currently ubiquitous, e.g., in genome sequences collected from similar species and in versioned documents maintained by Wikipedia and GitHub. Because the amount of data that includes repetitive strings is increasing rapidly, data processing methods based on grammar compression have been studied extensively as a promising way to address repetitive strings (e.g., [2]–[11]).

*Frequent pattern discovery* is a classic problem in pattern mining from sequence data (e.g., [12]), where we focus on a string and say that a pattern is frequent if it occurs at least twice. Longer patterns are often the target of discovery because they seem to better characterize the input string. We can solve this problem in linear time using the suffix array (SA) proposed in [13], however, SA requires a huge working space. Therefore, it is difficult to apply these algorithms to stream data. A reasonable approach to avoid this difficulty is to find an approximation of such a frequent pattern represented by a grammar. We consider that a variable $X$ of a grammar compression $G$ for a string $S$ approximates a substring $P$ of $S$ within approximation ratio $\delta$ iff for any interval $[i, j]$ with $P = S[i, j]$, the parse tree of $G$ has a node labeled with $X$ derives $S[\ell, r]$ for a subinterval $[\ell, r]$ of $[i, j]$ satisfying $|[\ell, r]| \geq \delta|[i, j]|$. Then, $G$ solves the frequent pattern discovery problem approximately within $\delta$ iff for any frequent pattern $P$ of $S$, there exists a variable that approximates $P$ within $\delta$. Here, $\delta$ is called the approximation ratio of $G$ for $S$. In this framework, an approximated frequent pattern is found as a frequent subtree in its parse tree. Then, a suitable parse tree should preserve as many occurrences of a common substring as possible. Edit-sensitive parsing (ESP) proposed in [14] satisfies this condition. ESP can solves the generalized edit distance problem approximately, which is known to be NP-hard. The generalized edit distance measures the similarity between two strings, and online algorithms and ESP applications have been proposed (e.g., [15]–[20]).

As seen above, the grammar compression problem is closely related to approximate pattern discovery because a good compression ratio is achieved by finding frequent substrings and replacing them with a variable that derives the substrings. In a previous study (e.g., [21]), ESP was applied to the approximate frequent pattern discovery. In that work, they proposed an offline algorithm that computes ESP as a grammar compression for an input string $S$ with the approximation ratio $\Omega(1/\lg^2|P|)$ for any substring $P$ in $S$. They also provided experimental results based on real data.

### 1.2 Contributions

We show a new lower bound $\Omega(\frac{1}{\lg^*|S|\lg|P|})$ for the approxima-

Manuscript received March 24, 2017.
Manuscript revised July 14, 2017.
Manuscript publicized December 19, 2017.

[†]The authors are with Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.

[*]A preliminary version of this paper was presented at ICGI2016 [1].

a) E-mail: hiroshi@ai.kyutech.ac.jp

DOI: 10.1587/transinf.2017FCP0010

tion ratio, which is an improvement of the previous bound $\Omega(\frac{1}{\lg^2 |P|})$ if $\lg^* |S| < \lg |P|$. Here, $\lg^* |S| = \min\{i \mid \lg^{(i)} |S| \leq 1\}$ for $\lg^{(1)} |S| = \lg |S|$ and $\lg^{(i+1)} |S| = \lg (\lg^{(i)} |S|)$. A certain type of frequent subtree in a grammar compression has been investigated relative to this result. Assuming a condition in the resulting ESP tree, we can easily show that the previous bound proved in [16] is transformed to the new lower bound. In this work, we remove this condition for any string.

We establish an online variant of previously proposed algorithm in [21] in compressed space. These algorithms are based on the compressed index presented in [16]. Note that the previous algorithms were not online, i.e., the entire string must be preloaded into memory. However, recent progress obtained in [9] has enabled the offline algorithm of [21] in compressed space in a streaming fashion. We implement our algorithm and show that the experimental value of the approximation ratio for real data is sufficiently larger than the theoretical bound.

## 2. Definitions

### 2.1 Notations

A sequence of symbols is called a *string*. The length of a string $S$ is denoted by $|S|$. The cardinality of a set $\mathcal{S}$ is also denoted by $|\mathcal{S}|$. For a fixed integer $\sigma$, $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$ is called an *alphabet*. Let $\Sigma^*$ be the set of all strings over $\Sigma$. An element of $\Sigma^q$ is called a *q-gram*. The *empty string* $\epsilon$ is a string of length 0. Let $\Sigma^+ = \Sigma^* - \{\epsilon\}$ and $a^{\geq 2} = \{a^k \mid k \geq 2\}$. A string in $a^{\geq 2}$ is called a *repetition* of $a$. For string $S = \alpha\beta\gamma$, $\alpha$, $\beta$ and $\gamma$ are called the *prefix*, *substring*, and *suffix* of $S$, respectively. The $i$-th character of string $S$ is denoted by $S[i]$ for $i \in [1, |S|]$. For a string $S$ and interval $[i, j]$ $(1 \leq i \leq j \leq |S|)$, let $S[i, j]$ denote the substring of $S$ that begins at position $i$ and ends at position $j$, and let $S[i, j]$ be $\epsilon$ when $i > j$. For a string $S$ and substring $P$ such that $S[i, j] = P$, the interval $[i, j]$ is called an occurrence of $P$ in $S$. For strings $S$ and $P$, let $freq_S(P)$ denote the number of occurrences of $P$ in $S$. Here, we assume a recursively enumerable set $\mathcal{X} = \{X_i \mid i = 1, 2, \ldots\}$ of variables with $\Sigma \cap \mathcal{X} = \emptyset$. All elements in $\Sigma \cup \mathcal{X}$ are totally ordered, where all elements in $\Sigma$ must be smaller than those in $\mathcal{X}$. Let $\lg^* N = \min\{i \mid \lg^{(i)} N \leq 1\}$ for $\lg^{(1)} N = \lg N$ and $\lg^{(i+1)} N = \lg (\lg^{(i)} N)$, e.g., $\lg^* N \leq 5$ for any $N \leq 2^{65536}$.

### 2.2 Grammar Compression

We consider a special type of CFG $G = (\Sigma, V, D, X_s)$ where $V = \{X_i \mid 1 \leq i \leq n\}$ is a finite subset of variables for some $n \geq 1$, $D$ is a finite subset of $V \times (V \cup \Sigma)^*$, and $X_s \in V$ is the start symbol. A *grammar compression* of a string $S$ is a CFG $G$ deriving only $S$ deterministically, i.e., $G$ derives $S$ such that for any $X \in V$, there exists exactly one *production rule* in $D$ and $D$ is *loop-free*: there is no variable that can derive itself by applying at least one production rule.

Because each $G$ has a Chomsky normal form, we can assume that any grammar compression is a *Straight-line program (SLP)* introduced in [22], i.e., any production rule is in the form of $X_k \to X_i X_j$ where $X_i, X_j \in \Sigma \cup V$ and $1 \leq i, j < k \leq n + \sigma$. The *size* of an SLP is $\sigma + n$.

For any $X \in V$, let $val(X)$ be the string $\alpha \in \Sigma^*$ derived from $X$. For any $a \in \Sigma$, let $val(a) = a$. For $w \in (V \cup \Sigma)^*$, let $val(w) = val(w[1]) \cdots val(w[|w|])$.

The *parse tree* of $G$ is a rooted ordered binary tree such that (i) the internal nodes are labeled by variables and (ii) the leaves are labeled by alphabet symbols. In a parse tree, any internal node $Z$ associated with $Z \to XY$ has left and right children with label $X$ and $Y$, respectively.

For a production rule $X \to \alpha \in D$, the variable $X$ and string $\alpha$ are called the *name* and *phrase* of the production rule, respectively. A *phrase dictionary D* is a data structure used to directly access the phrase for any given name. On the other hand, a *reverse dictionary* $D^{-1}$ is a data structure used to directly access the name for any given phrase if the corresponding production rule exists.

We define the problem of online grammar compression. For an input sequence of symbols, $a_1, a_2, \ldots$, let $G_i$ be a grammar compression for the string $a_1 a_2 \cdots a_i$. An online grammar compression algorithm is one that can compute $G_{i+1}$ from $(G_i, a_{i+1})$ without explicitly decompressing $G_i$.

### 2.3 Succinct Data Structure

A grammar compression is encoded using succinct data structures. A rank/select dictionary for a bit string $B$ proposed in [23] supports the following queries: $rank_c(B, i)$ returns the number of occurrences of $c \in \{0, 1\}$ in $B[1, i]$; $select_c(B, i)$ returns the beginning position of the $i$-th occurrence of $c \in \{0, 1\}$ in $B$; and $access(B, i)$ returns the $i$-th bit in $B$. Data structures with only $|B| + o(|B|)$ bits storage to achieve $O(1)$ time rank and select queries have been presented in [24]. The wavelet tree proposed in [25] is an extension of the rank/select dictionary to strings over $n$ different symbols for any $n \geq 2$. GMR presented in [26] is an improved wavelet tree in $n \lg n + o(n \lg n)$ bits that supports both rank and access queries in $O(\lg \lg n)$ time and select queries in $O(1)$ time.

### 2.4 Approximate Frequent Pattern

A substring $P = S[i, j]$ is said to be *frequent* if $freq_S(P) \geq 2$. We focus on an approximate solution of the problem to find all frequent patterns. The approximated problem is defined as follows.

**Problem 1:** Let $T$ be a parse tree of a grammar compression deriving $S \in \Sigma^*$. A symbol $X$ appearing in $T$ is called a *core* of a substring $P$ of $S$ iff for each occurrence $[i, j]$ of $P$, there exists a node with label $X$ in $T$ that derives substring $S[\ell, r]$ for a subinterval $[\ell, r]$ of $[i, j]$. Then, $P$ is said to be approximated by $X$ with $\delta$ if $\frac{|val(X)|}{|P|} \geq \delta$. The approximate frequent pattern problem (AFP) involves computing $T$ that guarantees a core $X$ of any frequent pattern $P$ in $S$ with an approximation ratio $\delta > 0$.

AFP is well-defined with a small $\delta$ because for any $S$ and its substring $P$, any alphabet symbol forming $P$ satisfies the condition with $\delta = 1/|P|$. An offline algorithm with approximation ratio $\Omega(\frac{1}{\lg^2 |P|})$ has been proposed in [21]. We aim at constructing the parse tree using an online algorithm in compressed space with sufficiently large $\delta$. In the proposed algorithm, a grammar compression is represented by the edit sensitive parsing (ESP) and succinctly encoded in a post-order SLP (POSLP). We review related techniques in the following sections.

## 2.5 Edit Sensitive Parsing

ESP introduced in [14] has been widely applied in data compression and information retrieval (e.g., [15], [17]–[20]). ESP is intended to efficiently construct a consistent parsing for common substrings as follows.

Let $T$ be a rooted ordered tree and $(u, v)$ be a pair of leaves in $T$. If there is no other leaf between $u$ and $v$, the pair $(u, v)$ is said to be *adjacent*, and if $u$ is the rightmost leaf of a subtree $x$ and $v$ is the leftmost leaf of a subtree $y$ in $T$, the pair $(x, y)$ is also said to be adjacent. Let $T$ be a parse tree of a string $S \in \Sigma^*$. A sequence $X_1, X_2, \ldots, X_q$ of symbols in $T$ is called a *decomposition* of $S$ if $x_i$ is a subtree in $T$ labeled by $X_i$ $(i = 1, 2, \ldots, q)$, each $(x_i, x_i + 1)$ is adjacent $(i = 1, 2, \ldots, q - 1)$, and $val(X_1 X_2 \cdots X_q) = S$.

For each substring $S[i, j]$, we can decompose the substring into a sequence of subtrees labeled by symbols $X_1, X_2, \ldots, X_q$. For each frequent $P$ (e.g., $S[i, j] = S[k, \ell] = P$), we can find a consistent decomposition for the occurrences by a trivial decomposition of $X_1 = S[i], X_2 = S[i + 1], \ldots, X_q = S[j]$. As shown in [21], the parse tree constructed by ESP (an ESP tree) guarantees better decomposition, i.e., a common $(X_1, X_2, \ldots, X_q)$ is embedded into any occurrence of $P$ with $q = \Omega(\frac{|P|}{\lg^2 |P|})$. We show another lower bound $\Omega(\frac{|P|}{\lg^* |S| \lg |P|})$.

If $T$ is an ESP tree for $S \in \Sigma^*$, any occurrence of $P$ in $S$ has a common decomposition $X_1, X_2, \ldots, X_q$ for some $1 \leq q \leq |P|$. Thus, we can employ each $X_i$ in $T$ as a necessary condition of an occurrence of $P$ in $S$. Using this fact, we can find an approximate pattern from the ESP tree. Using this result, we can efficiently compute a smaller grammar compression that is closely related to AFP.

In the following, we review the ESP algorithm proposed in [14]. This algorithm, referred to as ESP-comp, computes an SLP from an input string $S$. The ESP-comp algorithm (i) partitions $S$ into $s_1 s_2 \cdots s_\ell$ such that $2 \leq |s_i| \leq 3$ for each $1 \leq i \leq \ell$, (ii) if $|s_i| = 2$, generates the production rule $X \to AB$ and replace the $AB$ by $X$ (the binary tree $X(A, B)$ corresponding to the production rule is referred to as a 2-tree), and if $|s_i| = 3$, generates the production rules $Y \to AX$ and $X \to BC$ for $s_i = ABC$, and replaces $s_i$ with $Y$ (the binary tree $Y(A, X(B, C))$ corresponding to the two production rules is referred to as a 2-2-tree), and (iii) iterates this process until $S$ becomes a symbol. Finally, ESP-comp builds an SLP representing the string $S$.

We focus on how to determine the partition $S = s_1 s_2 \cdots s_\ell$. A repetition $S[i, j] \in X^{\geq 2}$ for some $X \in \Sigma \cup V$ is called to be *maximal* if $S[i - 1] \neq S[i] \neq S[j + 1]$. First, $S$ is uniquely partitioned to the form $w_1 x_1 w_2 x_2 \cdots w_k x_k w_{k+1}$ where each $x_i$ is a maximal repetition of a symbol in $\Sigma \cup V$, and each $w_i \in (\Sigma \cup V)^*$ contains no repetition. Then, each $x_i$ is called type1, each $w_i$ of length at least $2 \lg^* |S|$ is type2, and any remaining $w_i$ is type3. If $|w_i| = 1$ $(i \geq 2)$, $x_{i-1} w_i$ is renamed by $x_{i-1}$, and if $|w_1| = 1$, $w_1 x_1$ is renamed by $x_1$. Thus, if $|S| > 2$, each $x_i$ and $w_i$ is of length at least two.

Next, ESP-comp parses each substring $v$ depending on the type. For type1 and type3 strings, the algorithm performs the *left-aligned parsing* as follows. If $|v|$ is even, the algorithm builds a 2-tree from $v[2j - 1, 2j]$ for each $j \in \{1, 2, \ldots, |v|/2\}$; otherwise, the algorithm builds a 2-tree from $v[2j - 1, 2j]$ for each $j \in \{1, 2, \ldots, \lfloor (|v| - 3)/2 \rfloor\}$ and builds a 2-2-tree from the last trigram $v[|v| - 2, |v|]$. If $v$ is type2, the algorithm further partitions it into short substrings of length two or three using the following *alphabet reduction*.

**Alphabet reduction:** Given a type2 string $v$, consider $v[i]$ and $v[i - 1]$ as binary integers. Let $p_i$ be the position of the rightmost 1 in the binary string $v[i] \oplus v[i - 1]$ and let $bit(p, v[i])$ be the bit of $v[i]$ at the $p$-th position. Then, $L(v)[i] = 2p_i + bit(p_i, v[i])$ is defined for any $i \geq 2$. Because $v$ is repetition-free (i.e., type2), the label string $L(v)[2, |v|]$ is also type2. Assume that any symbol in $v$ is an integer in $\{0, \ldots, N\}$, and $L(v)[2, |v|]$ is a sequence of integers in $\{0, \ldots, 2 \lg N + 1\}^{|v|-1}$. If we apply this procedure $\lg^* N$ times, we obtain $L^*(v)[\lg^* N + 1, |v|]$ (a sequence of integers in $\{0, \ldots, 5\}^{|v| - \lg^* N}$), where $L^*(v)[1, \lg^* N]$ is undefined[†]. When $L^*(v)[i-1, i+1]$ is defined, $v[i]$ is called the *landmark* if $L^*(v)[i] > \max\{L^*(v)[i - 1], L^*(v)[i + 1]\}$.

The iteration of alphabet reduction transforms $v$ into $L^*(v)$ satisfying the condition that any substring of $L^*(v)[\lg^* N + 1, |v|]$ of length at least 12 contains at least one label $L^*(v)[i]$ such that $v[i]$ is a landmark because $L^*(v)[\lg^* N + 1, |v|] \in \{0, \ldots, 5\}^{|v| - \lg^* N - 1}$ is also type2. Using this characteristic, ESP-comp determines the bigram $v[i, i + 1]$ to be replaced by a 2-tree for each landmark $v[i]$, where any two landmarks are not adjacent. Then, the replacement is deterministic. After replacing all landmarks, any remaining maximal substring $s$ is replaced by the left-aligned parsing, where, if $|s| = 1$, it is attached to its left or right block. In summary, Theorem 1 holds for type2 strings:

**Theorem 1:** ([14]) For type2 string $v$, whether $v[i]$ is a landmark or not is determined by only $v[i - (\lg^* |S| + 5), i + 5]$. In addition there is at least one landmark for every $\lg^* |S| + 12$ positions in $v$.

An example of ESP of an input string is shown in Figs. 1-(i) and (ii). For type2 substring $v$ (Fig. 1-(i)), $v$ is parsed according to landmarks. For simplicity, landmarks

[†]The number of iterations of alphabet reduction should not be changed arbitrarily according to each $v$. Thus, $N$ is set in advance to be a sufficiently large integer, e.g., $N = \Theta(|S|)$.

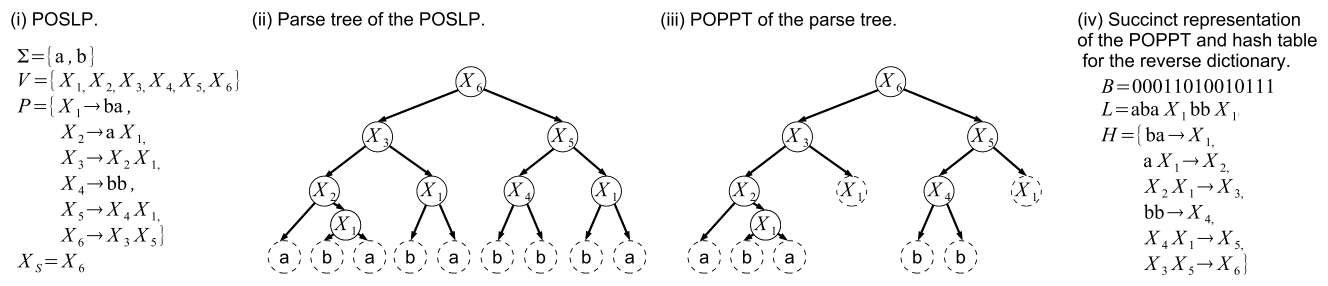

(i) POSLP.
Σ={a, b}
V={X1, X2, X3, X4, X5, X6}
P={X1→ba, X2→aX1, X3→X2X1, X4→bb, X5→X4X1, X6→X3X5}
XS=X6
(ii) Parse tree of the POSLP.
(iii) POPPT of the parse tree.
(iv) Succinct representation of the POPPT and hash table for the reverse dictionary.
B=00011010010111
L=aba X1 bb X1
H={ba→X1, aX1→X2, X2X1→X3, bb→X4, X4X1→X5, X3X5→X6}

**Fig. 2** Examples of POSLP, parse tree, post-order partial parse tree (POPPT), and a succinct representation of POPPT.

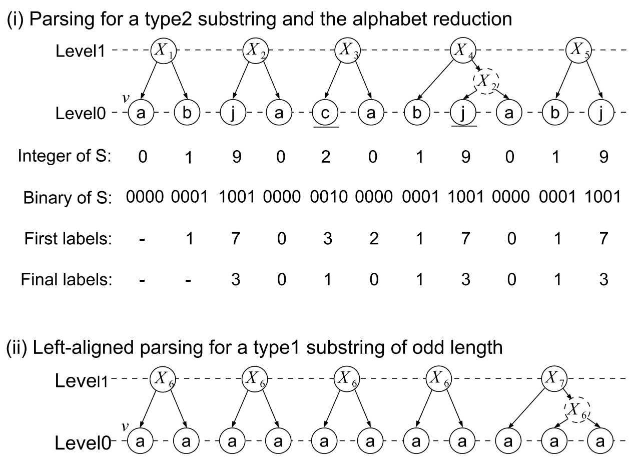

(i) Parsing for a type2 substring and the alphabet reduction
Level1
Level0
Integer of S: 0 1 9 0 2 0 1 9 0 1 9
Binary of S: 0000 0001 1001 0000 0010 0000 0001 1001 0000 0001 1001
First labels: - 1 7 0 3 2 1 7 0 1 7
Final labels: - - 3 0 1 0 1 3 0 1 3
(ii) Left-aligned parsing for a type1 substring of odd length
Level1
Level0

**Fig. 1** ESP. In (i), an underlined $v[i]$ means a landmark. In (i) and (ii), a dashed node corresponds to the intermediate node in a 2-2-tree.

are determined by performing alphabet reduction twice. Any other remaining substrings, including type1 and type3, are parsed by left-aligned parsing (Fig. 1-(ii)). In Fig. 1, a dashed node indicates an intermediate node in a 2-2-tree. Originally, an ESP tree is a ternary tree in which each node has at most three children. The intermediate node is introduced to represent an ESP tree as a binary tree.

Since a single ESP application shrinks the length of a string by a factor of at least two, the number of ESP iterations is bounded by $O(\lg|S|)$; thus, the following theorem holds.

**Theorem 2:** ([14]) The height of the ESP tree of $S$ is $O(\lg|S|)$.

## 2.6 Succinct Encoding

A partial parse tree introduced in [5] is a binary tree built by traversing a parse tree in a depth-first manner and pruning all descendants under every node of a variable that appears previously. Post-order SLP (POSLP) and post-order partial parse tree (POPPT) introduced in [8] are defined as follows.

**Definition 1** (POSLP and POPPT): A POSLP is an SLP whose parse tree's internal nodes have post-order variables. A POPPT is a partial parse tree of POSLP.

For a POSLP of $n$ variables, the number of nodes in the POPPT is $2n + 1$ because the numbers of internal nodes and leaves are $n$ and $n + 1$, respectively. Figures 2-(i) and (iii) show POSLP and POPPT examples, respectively. The resulting POPPT (Figs. 2 (iii)) has internal nodes consisting of post-order variables.

FOLCA proposed in [9] is a fully-online algorithm for computing succinct POSLP represented by $(B, L)$. Here, $B$ is the bit string obtained by traversing POPPT in post-order, and putting 0 if a node is a leaf and 1 otherwise. The last bit 1 in $B$ represents the super root. $L$ is the sequence of leaves of the POPPT. The dynamic sequences $B$ and $L$ are encoded using the succinct data structure proposed in [27]. Then, FOLCA achieved the following performance.

**Theorem 3** ([9]): A POSLP of $n$ variables and $\sigma$ alphabet symbols supporting the phrase and reverse dictionaries can be constructed in $O(\frac{|S|\lg n}{\alpha \lg\lg n})$ expected time using $(1+\alpha)n\lg(n+\sigma) + n(3+\lg(\alpha n))$ bits of memory, where $\alpha$ $(1/\alpha > 1)$ is the load factor of a hash table.

In this paper, we present a new lower bound for the approximation ratio of the AFP. Experimental results obtained with read data demonstrate that the expected ratio is sufficiently large. The implementation of the algorithm is realized by modifying FOLCA using novel data structures.

## 3. Algorithms

### 3.1 Offline Algorithm

Here, we review the offline construction of ESP presented in [16] applied to the offline AFP in [21]. Given a string $S \in \Sigma^*$, an AFP algorithm constructs an ESP tree $T$ for $S$ and finds all frequent variables as cores in $T$ that approximately represent all frequent patterns in $S$. Relative to a relation between pattern and its cores, the following result was shown.

**Theorem 4:** ([16]) Let $T$ be the ESP tree for string $S$. For any substring $P$ of $S$, there is a sequence $Q$ of cores with $val(Q) = P$ that can be decomposed as $Q = Q_1Q_2\cdots Q_k$, where each $Q_i$ is a sequence of cores of $P$ consisting of either a repetition of the form $Q_i \in c_i^{\geq 2}$ $(c_i \in \Sigma \cup V)$ or a string of length $O(\lg^*|S|)$ and $k = O(\lg|P|)$.

Here, we note that $k$ in Theorem 4 is not $|Q|$ itself.

When $|Q_i| = O(\lg^* |S|)$ for any $i$, the result of Theorem 4 is nearly equivalent to the result we attempt to prove in this study. To show the general case, we begin with the following outline of the algorithm proposed in [16] and its correctness.

Let $P = \alpha\beta\gamma$ where $\alpha$ is the first type string and $\gamma$ is the last type string. When $\alpha, \gamma \neq \epsilon$, the pair $(C, C')$ of strings of cores is defined as follows.

(1) If $\alpha$ is type2, $C = \alpha[1, i-1]$ where $i$ is the smallest integer satisfying the condition that $\alpha[i]$ is a landmark and $i > \lg^* |S| + 5$; otherwise, $C = \alpha$.
(2) If $\gamma$ is type2, $C' = \gamma[|\gamma| - j, |\gamma|]$ where $j$ is the smallest integer satisfying the condition that $\alpha[|\gamma| - j]$ is a landmark and $j > 5$; otherwise, $C' = \gamma$.
(3) For the determined partition $P = CP'C'$, let $P_1$ be the substring in the next level of the ESP deriving exactly $P'$. Continue the above process for $P'$ while $|P'| > 1$.

When $P$ is formed by a single type string i.e., $P = \alpha$, if $\alpha$ is type2, the pair $(C, C')$ is similarly defined by (1) and (2); otherwise, $(C, C') = (\alpha, \epsilon)$.

Let $(C_i, C'_i)$ be the pair obtained in the $i$-th iteration. It is guaranteed that any symbol forming $(C_i, C'_i)$ is a core of $P$ because the parsing of a type string $\alpha$ starts at the first symbol and whether $\alpha[i]$ is a landmark is determined by $\alpha[i - (\lg^* |S| + 5), i + 5]$ (Theorem 1). Note that $C_i$ is either a short string of length at most $O(\lg^* |S|)$ or a type1 string, as is $C'_i$, for each $i$. Moreover, the number of $(C_i, C'_i)$ is $O(\lg |P|)$ because $|P'| < |P|/2$ in each iteration. Thus, we obtain the sequence $Q = C_1 \cdots C_\ell C'_\ell \cdots C'_1 = Q_1 Q_2 \cdots Q_k$ for some $Q_i$ and $k = O(\lg |P|)$ in Theorem 4. We extend this theorem by deriving the new lower bound of a maximal core and propose an online algorithm to extract each $Q_i$.

## 3.2 OAFP Algorithm

We propose an online algorithm for AFP by a modification of FOLCA with a compressed space. We show the new lower bound of the size of the extracted core as well as the time and space complexities.

The proposed algorithm is summarized as follows. Let $S_i$ ($i = 0, 1, \ldots, \lceil \lg |S| \rceil$) be the resulting string of the $i$-th iteration of ESP, where $S_0 = S$. The algorithm simulates ESP using a queue $q_i$ for each level $i$. The queue $q_i$ stores a substring of $S_i$ of length at most $O(\lg^* |S|)$ in FIFO manner.

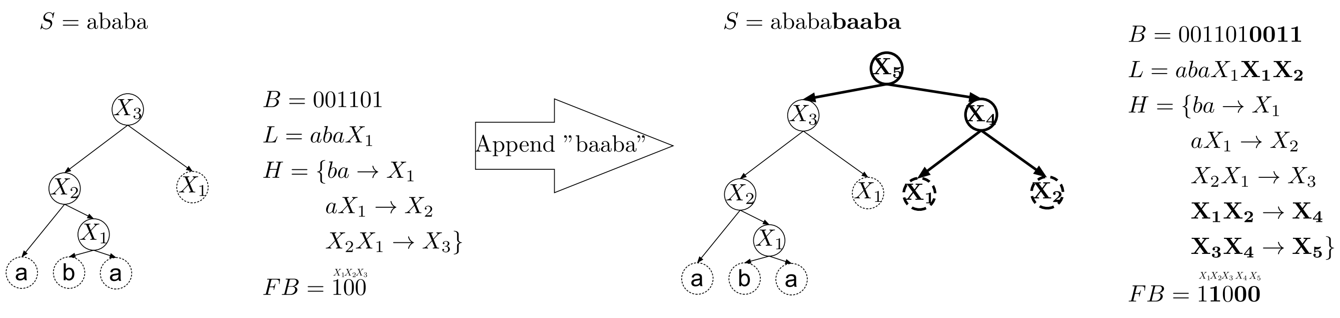
Each $S_i$ has its decomposition $S_i = \alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_n}$ where $\alpha_{i_\ell}$ is the $\ell$-th type string in $S_i$. If $\alpha_{i_\ell}$ is other than type2, it is parsed in left-aligned manner in $O(\lg^* |S|)$ space because we can determine the type of $\alpha_{i_\ell}$ in $O(\lg^* |S|)$ space. If $\alpha_{i_\ell}$ is type2, each block $\alpha_{i_\ell}[j, j' - 1]$ is parsed in left-aligned manner, where $j$ is the position of a landmark and $j'$ is the position of the next landmark. The space of $q_i$ for this case is also $O(\lg^* |S|)$ because we can decide whether $\alpha_{i_\ell}[j]$ is a landmark by at most $\alpha_{i_\ell}[j - (\lg^* |S| + 5), j + 5]$ (Theorem 1). After left-aligned parsing each case, the processed string is dequeued from $q_i$ and the generated string is enqueued in $q_{i+1}$. Here, note that $i = O(\lg |S|)$ because the number of queues is bounded by the height of the corresponding ESP tree. When a bigram $XY$ in $S_i$ is parsed by an implicit 2-tree $Z \to XY$, it is encoded in the corresponding POSLP $T$ represented by $(B, L)$, where $B$ is a bit sequence that represents the skeleton of $T$, and $L$ is the sequence of leaves of $T$. The pseudo code is shown in Algorithm 1.

**Algorithm 1** to compute core $X$ of any frequent $P$ in $S$. $T$: POSLP for the ESP tree of $S$; $B$: succinct representation of the skeleton of $T$; $L$: a sequence of leaves of $T$, $FB$: bit vector storing $FB[i] = 1$ iff $freq_T(X_i) \geq 2$; $D^{-1}$: reverse dictionary for production rules; $q_k$: queue in the $k$-th level.

```
1:  function ComputeAFP($S$)
2:      $B := \emptyset; L := \emptyset; FB := \emptyset$; initialize $q_k$; $u := \max\{5, \lg^* |S|\}$
3:      for $i := 1, 2, \ldots, |S|$ do
4:          BuildESPTree($(S[i], 0, 0, 0, 0), q_1$)
5:  function BuildESPTree($X, q_k$)
6:      ▷ $X$ is a tuple $(s, ib, \ell_1, \ell_2, \ell_{\lg^* |S|})$ where $s$ is a symbol, $ib$ is 1 if $s$ is
        an internal node otherwise 0 and $\ell_i (i \in \{1, 2, \lg^* |S|\})$ is a label applied
        $i$-th alphabet reduction for $s$.
7:      $q_k.enqueue(X)$
8:      compute $q_k[q_k.length()].\ell_i (i \in \{1, 2, \lg^* |S|\})$
9:      if $q_k.length() = u$ then
10:         if Is2Tree($q_k$) then
11:             $Y :=$ Update($q_k[u-1], q_k[u]$)
12:             $q_k.dequeue(); q_k.dequeue()$
13:             BuildESPTree($Y, q_{k+1}$)
14:     else if $q_k.length() = u + 1$ then
15:         $Y :=$ Update($q_k[u], q_k[u+1]$); $Z :=$ Update($q_k[u-1], Y$)
16:         $q_k.dequeue(); q_k.dequeue(); q_k.dequeue()$
17:         BuildESPTree($Z, q_{k+1}$)
18: function Is2Tree($q_k$)
19:     if $(q_k[u-4].s = q_k[u-3].s) \& (q_k[u-3].s \neq q_k[u-2].s)$ then
20:         return 0
21:     else if $(q_k[u-3].s \neq q_k[u-2].s) \& (q_k[u-2].s = q_k[u-1].s)$ then
22:         return 0
23:     else if $(q_k[u-3].\ell_{\lg^* |S|} < q_k[u-2].\ell_{\lg^* |S|}) \& (q_k[u-2].\ell_{\lg^* |S|} >$
        $q_k[u-1].\ell_{\lg^* |S|})$ then
24:         return 0
25:     else
26:         return 1
27: function Update($X, Y$)
28:     $z := D^{-1}(X.s, Y.s)$
29:     if $z$ is a new symbol then
30:         UpdateLeaf($X$); UpdateLeaf($Y$)
31:         $B.push\_back(1); FB.push\_back(0)$
32:         return $(z, 1, 0, 0, 0)$
33:     else
34:         GetAFPNode($z$)
35:         return $(z, 0, 0, 0, 0)$
36: function UpdateLeaf($X$)
37:     if $X.ib = 0$ then
38:         $L.push\_back(X.s); B.push\_back(0)$
39: function GetAFPNode($X_i$)
40:     if $FB[i] = 0$ then
41:         $FB[i] := 1$
42:         Output $X_i$
```

We next show that the ESP tree of $S$ contains a sufficiently large core for any substring $P$ that guarantees the approximation ratio of our algorithm.

**Theorem 5:** Let $T$ be the ESP tree of a string $S$ and $P$ be a substring of $S$. There exists a core of $P$ that derives a string of length $\Omega(\frac{|P|}{\lg^* |S| \lg |P|})$.



$S$ = ababa
$B = 001101$
$L = abaX_1$
$H = \{ba \to X_1$
$aX_1 \to X_2$
$X_2X_1 \to X_3\}$
$FB = \overset{X_1X_2X_3}{100}$
Append "baaba"
$S$ = ababa**baaba**
$B = 001101\mathbf{0011}$
$L = abaX_1\mathbf{X_1X_2}$
$H = \{ba \to X_1$
$aX_1 \to X_2$
$X_2X_1 \to X_3$
$\mathbf{X_1X_2 \to X_4}$
$\mathbf{X_3X_4 \to X_5}\}$
$FB = \overset{X_1X_2X_3X_4X_5}{11\mathbf{000}}$

**Fig. 3** The flow of the proposed online algorithm. For the input string $S$ = *ababa* received so far, the illustrated SLP is directly encoded in $B$ = 0001101 and $L = abaX_1$. Here, $FB[i] = 1$ iff $X_i$ is frequent, and $H$ is the hash function for naming of variables. When the string *baaba* is appended to $S$, the algorithm updates $B$, $L$, and $FB$ for the corresponding SLP.

**proof.** By Theorem 4, we can obtain a sequence $Q_1Q_2\cdots Q_k$ such that each $Q_i$ is a sequence of cores of $P$ consisting of either a repetition of the form $Q_i \in c_i^{\geq 2}$ ($c_i \in \Sigma \cup V$) or a string of length $O(\lg^* |S|)$.

We show that for any $1 \leq i \leq k$, there exists a core $X_i$ in $Q_i$ with $|val(X_i)| = \Omega(\frac{|val(Q_i)|}{\lg^* |S|})$. If the length of $Q_i$ is $O(\lg^* |S|)$, the claim is immediate according to the pigeonhole principle. Otherwise, $Q_i \in c_i^{\geq 2}$. Any maximal repetition is parsed in left-aligned manner; thus, a type2 sequence of bigrams $c_i^2$ is created over $Q_i$ (except for the last one, which may be a 2-2-tree deriving $c_i^3$). Iterating the parsing of the type2 sequence, we obtain a large and complete balanced binary tree of $c_i$. Assuming that a largest core covers $2^h$ $c_i$'s in $Q_i$, we observe that the number of $c_i$'s in $Q_i$ is less than $5 \cdot 2^h$, i.e., there is a node that covers at least one-fifth of the $c_i$'s in $Q_i$. The maximum length of $Q_i$ is achieved when $Q_i$ is parsed into $ABC_{h-1}\cdots C_0$, where $A$ contains $2^h - 1$ $c_i$'s, $B$ contains $2^h$ $c_i$'s, and, for any $0 \leq h' < h$, $C_{h'}$ contains $3 \cdot 2^{h'}$ $c_i$'s. $A$ and its preceding character $c \neq c_i$ (that must be the first character in the entire string) form a node with $2^h$ characters, $B$ composes the largest complete binary tree with $2^h$ $c_i$'s, and, for any $0 \leq h' < h$, $C_{h'}$ forms a 2-2-tree over three complete binary trees with $2^{h'}$ $c_i$'s. Adding even a single $c_i$ to $Q_i$ results in creating a complete binary tree with $2^{h+1}$ $c_i$'s (which may appear in a 2-2-tree over three complete binary trees with $2^h$ $c_i$'s); thus, the maximum number of $c_i$'s in $Q_i$ is $2^h - 1 + 2^h + \sum_{h'=0}^{h-1} 3 \cdot 2^{h'} < 5 \cdot 2^h$. Therefore, there exists a variable $X_i$ in $Q_i$ with $|val(X_i)| = \Omega(\frac{|val(Q_i)|}{\lg^* |S|})$.

There is at least one $Q_j$ such that $|val(Q_j)| \geq |P|/k \geq |P|/\lg|P|$; therefore, there exists a core of $P$ that derives a string of length $\Omega(\frac{|val(Q_j)|}{\lg^* |S|}) = \Omega(\frac{|P|}{\lg^* |S| \lg |P|})$. □

**Theorem 6:** Algorithm 1 solves the AFP problem approximately at a ratio $\delta = \Omega(\frac{1}{\lg^* |S| \lg |P|})$ in $O(\frac{|S| \lg n}{\alpha \lg \lg n})$ expected time and $O(n + \lg |S|)$ space.

**proof.** The algorithm simulates the ESP of $S$ using queues $q_i$ ($i = 0, 1, \ldots, |S|$). Here, $q_i$ stores a substring of $S_i$ to determine whether $S_i[j]$ is a landmark. According to Theorem 1, the space occupied by each $q_i$ is $O(\lg^* |S|)$. We can reduce this space to $O(1)$ using a table of size at most $\lg^* |S| \lg\lg\lg |S|$ bits as follows. By applying two iterations of alphabet reduction, each symbol $A$ is transformed into a label $L_A$ of size at most $\lg\lg\lg |S|$ bits. Whether the $A$ is a landmark or not depends on its consecutive $O(\lg^* |S|)$ neighbors. Thus, the size of a table storing a 1-bit answer is at most $\lg^* |S| \lg\lg\lg |S|$ bits. It follows that the space for parsing $S$ is $O(\lg |S|)$. On the other hand, according to Theorem 3, the POSLP $T$ of $S$ is computable in $O(\frac{|S| \lg n}{\alpha \lg \lg n})$ expected time. By Theorem 5, for each frequent $P$, $T$ contains at least one core $X$ of $P$ satisfying $|val(X)| = \Omega(\frac{|P|}{\lg^* |S| \lg |P|})$. Thus, finding all variables $X$ appearing at least twice in $T$ solves AFP problem approximately with the lower bound. Whether $freq_T(X_i) \geq 2$ can be stored in $n$ bits for all $i$ because an internal node $i$ of $T$ indicates the position of the first occurrence of $X_i$. Therefore, we obtain the complexities and approximation ratio. □

By Theorem 5, any pattern $P$ in $S$ is approximated by a variable in ESP tree of $S$ with the ratio. By Theorem 6, AFP is online computable in a compressed space. Next, we demonstrate the proposed algorithm with real data.

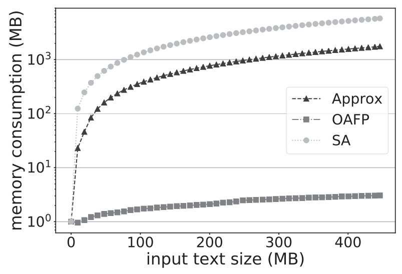
## 4. Experimental Results

We evaluated the performance of the proposed online AFP (OAFP) algorithm on one core of a quad-core Intel Xeon Processor E5540 (2.53GHz) machine with 144GB memory. We adopted a lightweight version of fully-online ESP, i.e., FOLCA proposed in [9], as a subroutine for grammar compression algorithm.

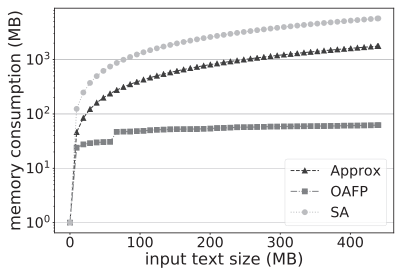
We used several benchmarks from a text collection (http://pizzachili.dcc.uchile.cl/repcorpus.html). Moreover, we use text collections from Wikipedia dump files (https://dumps.wikimedia.org) and source programs from GitHub (https://github.com). For these texts, we examined the practical approximation ratio of the algorithm as follows. For each text $S$, we obtained the set of frequent substrings by the compressed SA proposed in [13], and we selected the top-100 longest patterns so that any two $P$ and $Q$ are not *dominant* over each other, where a substring $P$ is considered dominant over $Q$ if, for any occurrence $[i, j]$ of $Q$, there exists an occurrence $[\ell, r]$ of $P$ such that $[\ell, r]$ is a subinterval of $[i, j]$. We removed such $Q$ from the candidates. For each

**Table 1** SA and OAFP (max.) is the length of longest frequent pattern in top-100 extracted by SA and the proposed OAFP algorithm, respectively, and SA and OAFP (min./mean) are analogous. Approx (max.) is the largest approximation ratio $\frac{|val(Q_j)|}{|P_i|}$ (%) such that $Q_j$ is a core of $P_i$, and Approx (min./mean) are analogous.

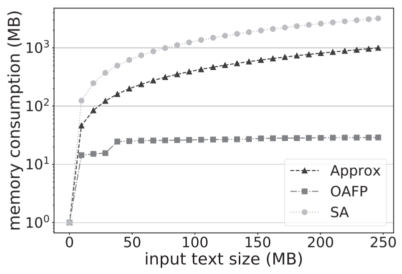
| data (MB) | | **einstein** 446 | **cere** 446 | **kernel** 246 | **english** 200 | **dna** 200 | **sources** 200 | **wikipeida** 5,523 | **GitHub** 8,180 |
|---|---|---|---|---|---|---|---|---|---|
| max. | SA | 935, 920 | 303, 204 | 2, 755, 550 | 98, 7770 | 97, 979 | 307, 871 | N/A | N/A |
| | OAFP | 342, 136 | 58, 906 | 662, 630 | 16, 1320 | 24, 834 | 57, 508 | 855262 | 6724671 |
| | Approx | **50.0** | **62.1** | **52.8** | **50.8** | **63.9** | **51.7** | - | - |
| min. | SA | 198, 606 | 4, 562 | 442, 124 | 43, 985 | 3, 271 | 4, 776 | N/A | N/A |
| | OAFP | 18, 625 | 4, 096 | 37, 205 | 3, 382 | 268 | 477 | 4096 | 11396 |
| | Approx | **7.6** | **2.3** | **6.9** | **7.3** | **7.1** | **7.3** | - | - |
| mean | SA | 259, 451 | 111, 284 | 727, 443 | 116, 920 | 8, 241 | 14, 498 | N/A | N/A |
| | OAFP | 56, 584 | 12, 723 | 152, 903 | 24, 703 | 1, 926 | 3, 279 | 33345 | 133654 |
| | Approx | **21.6** | **11.0** | **20.0** | **23.0** | **22.9** | **22.0** | - | - |



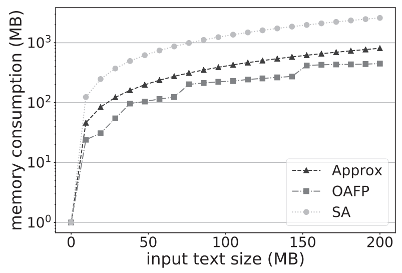
(a) einstein



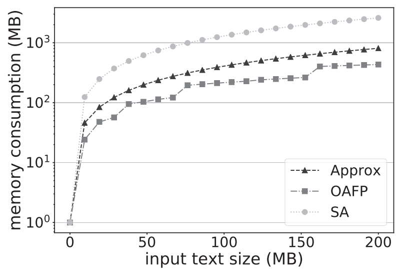
(b) cere



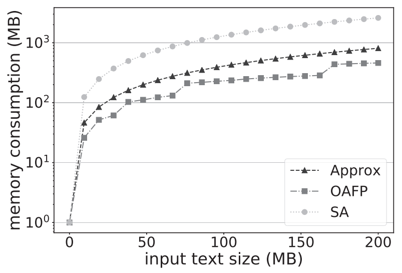
(c) kernel



(d) english



(e) dna



(f) sources

**Fig. 4** Memory consumption (MB)

frequent substring $P$ and variable $X$ reported by the algorithm, we estimated the cover ratio $\frac{|val(X)|}{|P|}$: Let $P_i$ and $Q_j$ be one of the top-100 patterns extracted by the SA and the proposed OAFP algorithm, respectively. When $X_i$ is a core of $P_i$ such that $val(X_i) = Q_i$, the approximation ratio $\frac{|val(X_i)|}{|P_i|}$ was evaluated. However, as shown in the result below (Fig. 4), the SA cannot be executed for larger $S$ due to memory consumption. In addition, we examined the time and memory consumption of the offline AFP algorithm proposed in [21].

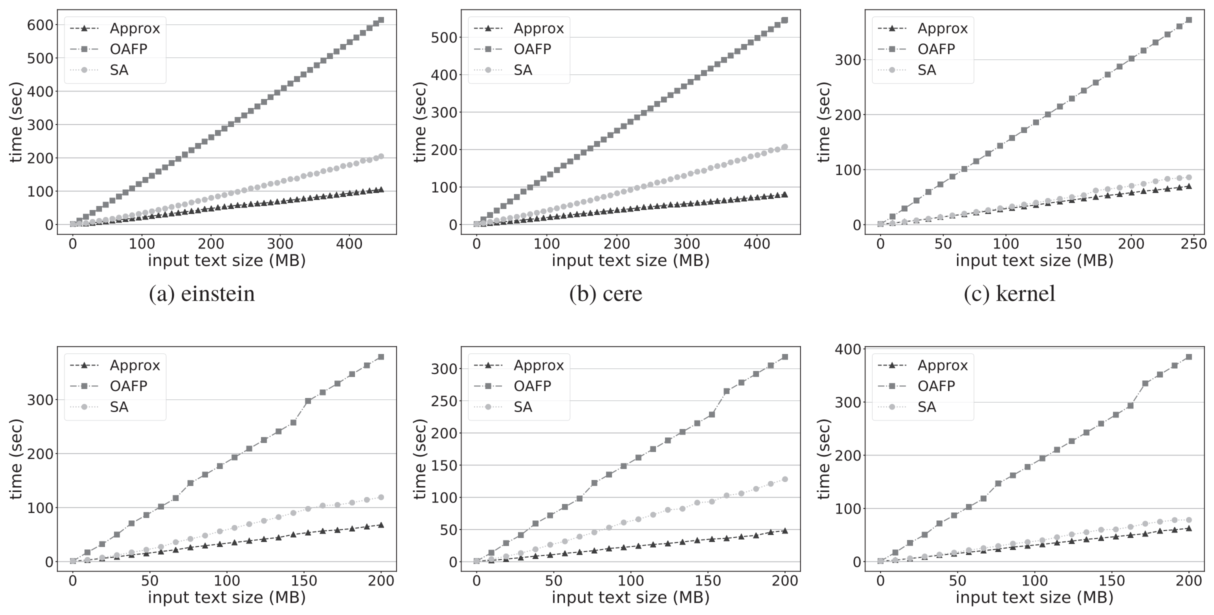
Table 1 shows a summary of the longest top-100 frequent patterns extracted by the SA and the proposed OAFP algorithm using benchmark data from various domains. We also show the approximation ratio $\frac{|val(X_i)|}{|P_i|}$ for the pattern $P_i$ extracted by the SA and a corresponding core found by OAFP algorithm. For example, the length of a longest frequent pattern $P$ in **einstein** was 935,920 and the corresponding length of $Q$ obtained by OAFP algorithm is 343,136. The approximation ratio in this case was not the ratio of $P$ and $Q$ because this $Q$ is a substring of other frequent pattern in this case. We evaluated the exact ratio according to the definition of the extracted patterns. As a result, the proposed OAFP algorithm extracted sufficiently large cores for each benchmark because $|Q|/|P|$ was considerably larger than the theoretical bound $\frac{1}{\lg^* |S| \lg |P|}$.

Figure 4 shows the memory consumption for repetitive strings (Figs. 4 (a)–4 (c)) and normal strings (Figs. 4 (d)–4 (f)). The working space was significantly saved by our online strategy, whereas offline and SA algorithms were executed for each static size of data noted in the figures.

Figure 5 shows the computation time for each benchmark. Due to the time-space tradeoff of a succinct data structure, the proposed algorithm was 2.48–6.82 times slower than the SA and the offline algorithm.

## 5. Conclusion and Future Work

We have proposed an online approximation algorithm in compressed space for the problem of finding frequent pat-



(a) einstein (b) cere (c) kernel

(d) english (e) dna (f) sources

**Fig. 5** Time consumption (sec)

terns. The proposed algorithm is an improvement of the FOLCA algorithm, which is a fully-online grammar compression algorithm. We derived a new theoretical lower bound of the approximation ratio, that is larger than the previous bound under an assumption of input string, and presented experimental results that demonstrate efficiency for highly repetitive texts. The new lower bound holds on the previous offline algorithms in [16] and [21] because the proposed online algorithm simulates the offline construction of ESP tree. However, relative to the approximation ratio, a large gap between theory and practical results. Thus, improving the lower bound is important future work.

Another approach to solve the problem in compressed space is to use the run-length encoded Burrows-Wheeler Transform (RLBWT) of a string $S$, which takes space linear to the number $r$ of runs in the BWT of $S$ of length $n$. Given the RLBWT of $S$, we can simulate the enumeration of all nodes of the suffix tree of $S$ (e.g., Sect. 9.4.1 in [28]) and thus solve the frequent pattern discovery problem exactly. Although there is an implementation of the online construction of RLBWT that runs in $O(n \lg r)$ time and $O(r \lg n)$ space proposed in [29], preliminary experiments showed that its execution time is slow compared to our proposed approach. Therefore, in future, it would be interesting to engineer the RLBWT approach to realize faster execution times.

## Acknowledgments

This research was supported by JST CREST (Grant Number JPMJCR1402), and KAKENHI (Grant Numbers 17H01791 and 16K16009). The authors thank ICGI2016 and IEICE reviewers for their careful comments.

## References

[1] S. Fukunaga, Y. Takabatake, T.I, and H. Sakamoto, "Online grammar compression for frequent pattern discovery," ICGI2016, pp.93–104. 2016.
[2] N.J. Larsson and A. Moffat, "Off-line dictionary-based compression," Proc. IEEE, vol.88, no.11, pp.1722–1732, 2000.
[3] E. Lehman and A. Shelat, "Approximation algorithms for grammar-based compression," SODA, pp.205–212, 2002.
[4] E.-H. Yang and D.-K. He, "Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform - part two: with context models," IEEE Trans. Inf. Theory, vol.49, no.11, pp.2874–2894, 2003.
[5] W. Rytter, "Application of Lempel—Ziv factorization to the approximation of grammar-based compression," Theor. Comput. Sci., vol.302, no.1-3, pp.211–222, 2003.
[6] H. Sakamoto, "A fully linear-time approximation algorithm for grammar-based compression," J. Discrete Algorithms, vol.3, no.2-4, pp.416–430, 2005.
[7] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat, "The smallest grammar problem," IEEE Trans. Inf. Theory, vol.51, no.7, pp.2554–2576, 2005.
[8] S. Maruyama, H. Sakamoto, and M. Takeda, "An online algorithm for lightweight grammar-based compression," Algorithms, vol.5, no.4, pp.213–235, 2012.
[9] S. Maruyama, Y. Tabei, H. Sakamoto, and K. Sadakane, "Fully-online grammar compression," SPIRE, vol.8214, pp.218–229, 2013.
[10] Y. Tabei, Y. Takabatake, and H. Sakamoto, "A succinct grammar compression," CPM, vol.7922, pp.218–229, 2013.
[11] S. Maruyama and Y. Tabei, "Fully online grammar compression in constant space," DCC, pp.218–229, 2014.
[12] C. Aggarwal and J. Han, eds., Frequent Pattern Mining, Springer, 2014.
[13] K. Sadakane, "Compressed text databases with efficient query algorithms based on the compressed suffix array," ISAAC, vol.1969, pp.410–421, 2000.
[14] G. Cormode and S. Muthukrishnan, "The string edit distance matching problem with moves," ACM Trans. Algorithms, vol.3, no.1,

pp.1–19, 2007.

[15] F. Hach, I. Numanagić, C. Alkan, and S.C. Sahinalp, "Scalce: boosting sequence compression algorithms using locally consistent encoding," Bioinformatics, vol.28, no.23, pp.3051–3057, 2012.

[16] S. Maruyama, M. Nakahara, N. Kishiue, and H. Sakamoto, "ESP-Index: A compressed index based on edit-sensitive parsing," J. Discrete Alogrithms, vol.18, pp.100–112, 2013.

[17] Y. Takabatake, Y. Tabei, and H. Sakamoto, "Online pattern matching for string edit distance with moves," SPIRE, vol.8799, pp.203–214, 2014.

[18] Y. Takabatake, Y. Tabei, and H. Sakamoto, "Online self-indexed grammar compression," SPIRE, vol.9309, pp.258–269, 2015.

[19] Y. Takabatake, K. Nakashima, T. Kuboyama, Y. Tabei, and H. Sakamoto, "siEDM: an efficient string index and search algorithm for edit distance with moves," Algorithms, vol.9, no.2, p.Article 26, 2016.

[20] T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda, "Dynamic index, lz factorization, and lcequeries in compressed space," arXiv: CoRR abs/1504.06954, 2015.

[21] M. Nakahara, S. Maruyama, T. Kuboyama, and H. Sakamoto, "Scalable detection of frequent substrings by grammar-based compression," IEICE Trans. Inf. Syst, vol.E96-D, no.3, pp.457–464, 2013.

[22] M. Karpinski, W. Rytter, and A. Shinohara, "An efficient pattern-matching algorithm for strings with short descriptions," Nord. J. Comput., vol.4, pp.172–186, 1997.

[23] G. Jacobson, "Space-efficient static trees and graphs," Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS), Research Triangle Park, North Carolina, USA, 30 October-1 November, pp.549–554, 1989.

[24] R. Raman, V. Raman, and S.R. Satti, "Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets," ACM Trans. Algor., vol.3, no.4, p.43, 2007.

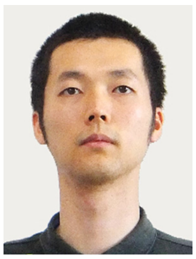
[25] R. Grossi, A. Gupta, and J. Vitter, "High-order entropy-compressed text indexes," SODA, pp.636–645, 2003.

[26] A. Golynski, J.I. Munro, and S.S. Rao, "Rank/select operations on large alphabets: A tool for text indexing," Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Miami, Florida, USA, pp.368–373, 22–26 January, 2006.

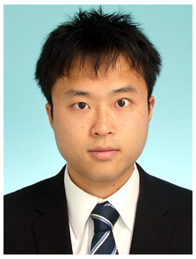
[27] G. Navarro and K. Sadakane, "Fully-functional static and dynamic succinct trees," ACM Trans. Algorithms, vol.10, no.3, pp.1–39, 2014.

[28] V. Mäkinen, D. Belazzougui, F. Cunial, and A.I. Tomescu, Genome-Scale Algorithm Design, Cambridge University Press, 2015.

[29] https://github.com/nicolaprezza/slz-rlbwt/tree/master/extern/DYNAMIC.



**Shouhei Fukunaga** received the B.S degree in Computer Science and Systems Engineering from Kyushu Institute of Technology in 2016. He is a student in master's course at Kyushu Institute of Technology.



**Yoshimasa Takabatake** received the B.Sci., M.Sci. and Dr.Sci. degrees in Computer Science and Systems Engineering from Kyushu Institute of Technology in 2012, 2014 and 2017, respectively. From Apr. 2015 to Mar. 2017, he was a JSPS Research Fellow (DC2). From Apr. 2017, he is currently a research associate at Kyushu Institute of Technology.

**Tomohiro I** received the B.Eng., M.Sci. and Dr.Sci. all from Kyushu University in Mar. 2008, Mar. 2010, and Apr. 2012, respectively. From May 2012 to Mar. 2014, he was a research fellow of JSPS Research Fellowship for Young Scientists (PD). From Apr. 2014 to Sept. 2015, he was a postdoctoral researcher at TU Dortmund, Germany. Since Oct. 2015, he has worked as an assistant professor at Kyushu Institute of Technology.

**Hiroshi Sakamoto** received B.S. degree in Physics and M.Sci. and Dr. Sci. degrees from Kyushu University in 1994, 1996, and 1998, respectively. He received JSPS Research Fellowships for Young Scientists from 1996 to 1998. From Jan. 1999 to Oct. 2003, he was a Research Associate in Department of Informatics, Kyushu University. From Nov. 2003 to Mar 2014, he was an Associate Professor in Department of Artificial Intelligence, Kyushu Institute of Technology, where he is currently a Professor. From Oct. 2009, he is concurrently JST PRESTO Researcher. He is also a member of IEICE, JSAI, and DBSJ.