A Succinct Grammar Compression*

Yasuo Tabei[1], Yoshimasa Takabatake[2], and Hiroshi Sakamoto[2,3]

[1] ERATO Minato Project, JST, Japan
[2] Kyushu Institute of Technology, Japan
[3] PRESTO, JST, Japan
tabei.y.aa@m.titech.ac.jp, {takabatake, hiroshi}@donald.ai.kyutech.ac.jp

Abstract. We solve an open problem related to an optimal encoding of a *straight line program* (SLP), a canonical form of grammar compression deriving a single string deterministically. We show that an information-theoretic lower bound for representing an SLP with n symbols requires at least $2n + \log n! + o(n)$ bits. We then present a succinct representation of an SLP; this representation is asymptotically equivalent to the lower bound. The space is at most $2n \log \rho(1 + o(1))$ bits for $\rho \leq 2\sqrt{n}$, while supporting random access to any production rule of an SLP in $O(\log \log n)$ time. In addition, we present a novel dynamic data structure associating a digram with a unique symbol. Such a data structure is called a *naming function* and has been implemented using a hash table that has a space-time tradeoff. Thus, the memory space is mainly occupied by the hash table during the development of production rules. Alternatively, we build a dynamic data structure for the naming function by leveraging the idea behind the *wavelet tree*. The space is strictly bounded by $2n \log n(1 + o(1))$ bits, while supporting $O(\log n)$ query and update time.

1 Introduction

Grammar compression has been an active research area since at least the seventies. The problem consists of two phases: (i) building the smallest† context-free grammar (CFG) generating an input string uniquely and (ii) encoding an obtained CFG as compactly as possible.

The phase (i) is known as an NP-hard problem which can not be approximated within a constant factor [21]. Therefore, many researchers have made considerable efforts to design grammar compressions achieving better approximation results in the last decade. Charikar et al. [6] and Rytter [29] independently proposed the first $O(\log \frac{u}{g})$-approximation algorithms based on balanced grammar construction for the length u of a string and the size g of the smallest CFG. Later, Sakamoto [31] also developed an $O(\log \frac{u}{g})$-approximation algorithm based on an idea called pairwise comparison. In particular, Lehman [21] proved

* This study was supported by KAKENHI(23680016,24700140) and JST PRESTO program.

† This is almost equal to minimizing the number of variables in G.

that LZ77 [35] achieved the best approximation of $O(\log n)$ under the condition of an unlimited window size. Since the minimum *addition chain* problem is a special case of the problem of finding the smallest CFG [20], modifying the approximation algorithms proposed so far is a difficult problem. Thus, the problem of grammar compression is pressing in the phase (ii).

A straight line program (SLP) is a canonical form of a CFG, and has been used in many grammar compression algorithms [36,19,35,1,23]. The production rules in SLPs are in Chomsky normal form where the right hand side of a production rule in CFGs is a *digram*: a pair of symbols. Thus, if n symbols are stored in an array called a *phrase dictionary* consisting of $2n$ fixed-length codes each of which is represented by $\log n$ bits, the memory of the dictionary is $2n \log n$ bits, resulting in the memory for storing an input string usually being exceeded. Although directly addressable codes achieving entropy bounds on strings whose memory consumption is the same as that of the fixed-length codes in the worst case have been presented [8,30,11], there are no codes that achieve an information-theoretic lower bound of storing an SLP in a phrase dictionary. Since a nontrivial information-theoretic lower bound of directly addressable codes for a phrase dictionary remains unknown, establishing the lower bound and developing novel codes for optimally representing an SLP are challenges.

We present an optimal and directly addressable SLP within a strictly bounded memory close to the amount of a plain representation of the phrase dictionary. We first give an information-theoretic lower bound on the problem of encoding an SLP, which has been unknown thus far. Let C be a class of objects. Representing an object $c \in C$ requires at least $\log |C|$ bits. A representation of c is succinct if it requires at most $\log |C|(1+o(1))$ bits. Considering the facts and the characteristics of SLPs indicated in [23], one can predict that the lower bound for the class of SLPs with n symbols would be between $2n$ and $4n + \log n!$. By leveraging this prediction, we derive that a lower bound of bits to represent SLPs is $2n + \log n!$.

We then present an almost optimal encoding of SLPs based on *monotonic subsequence decomposition* of a sequence. Any permutation of $[1, n]$ is decomposable into at most $\rho \leq 2\sqrt{n}$ monotonic subsequences in $O(n^{1.5})$ time [33] and there is a 1.71-approximation[‡] algorithm in $O(n^3)$ time [9]. While the previous encoding method for SLPs presented in [32] is also based on the decomposition, the size is not asymptotically equal to the lower bound when $\rho \simeq \sqrt{n}$. We improve the data structure by using the *wavelet tree* (WT) [12] and its improved results [3,10] such that our novel data structure achieves the smaller bound of $\min\{2n+n \log n+o(1), 2n \log \rho(1+o(1))\}$ bits for any SLP with n symbols while supporting $O(\log \log \rho)$ access time. Our method is applicable to any types of algorithm generating SLPs including Re-Pair [19] and an online algorithm called LCA [24]. Barbay et al. [4] presented a succinct representation of a sequence using the monotonic subsequence decomposition. Their method uses the representation of an integer function built on a succinct representation of integer ranges. Its size is estimated to be the *degree entropy* of an ordered tree [14].

[‡] Minimizing ρ is NP-hard.

Another contribution of this paper is to present a dynamic data structure for checking whether or not a production rule in a CFG has been generated in execution. Such a data structure is called a *naming function*, and is also necessary for practical grammar compressions. When the set of symbols is static, we can construct a perfect hash as a naming function in linear time, which achieves an amount of space within around a factor of 2 from the information-theoretical minimum [5]. However, variables of SLPs are generated step by step in grammar compression. While the function can be dynamically computed by a randomization [17] or a deterministic solution [16] in $O(1)$ time and linear space, a hidden constant in the required space was not clear. We present a dynamic data structure to compute function values in $O(\log n)$ query time and update time. The space is strictly bounded by $2n \log n(1 + o(1))$ bits.

2 Preliminaries

2.1 Grammar compression

For a finite set C, $|C|$ denotes its cardinality. *Alphabet* Σ is a finite set of letters and $\sigma = |\Sigma|$ is a constant. $\mathcal{X}$ is a recursively enumerable set of *variables* with $\Sigma \cap \mathcal{X} = \emptyset$. A sequence of symbols from $\Sigma \cup \mathcal{X}$ is called a string. The set of all possible strings from Σ is denoted by Σ^*. For a string S, the expressions $|S|$, $S[i]$, and $S[i, j]$ denote the length of S, the i-th symbol of S, and the substring of S from $S[i]$ to $S[j]$, respectively. Let $[S]$ be the set of symbols composing S. A string of length two is called a *digram*.

A CFG is represented by $\mathcal{G} = (\Sigma, V, P, X_s)$ where V is a finite subset of $\mathcal{X}$, P is a finite subset of $V \times (V \cup \mathcal{X})^*$, and $X_s \in V$. A member of P is called a production rule and X_s is called the start symbol. The set of strings in Σ^* derived from X_s by $\mathcal{G}$ is denoted by $L(\mathcal{G})$.

A CFG $\mathcal{G}$ is called *admissible* if exactly one $X \to \alpha \in P$ exists and $|L(\mathcal{G})| = 1$. An admissible $\mathcal{G}$ deriving S is called a grammar compression of S for any $X \in V$.

We consider only the case $|\alpha| = 2$ for any production rule $X \to \alpha$ because any grammar compression with n variables can be transformed into such a restricted CFG with at most $2n$ variables. Moreover, this restriction is useful for practical applications of compression algorithms, e.g., LZ78 [36], REPAIR [19], and LCA [24], and indices, e.g., SLP [7] and ESP [22].

The derivation tree of G is represented by a rooted ordered binary tree such that internal nodes are labeled by variables in V and the *yields*, i.e., the sequence of labels of leaves is equal to S. In this tree, any internal node $Z \in V$ has a left child labeled X and a right child labeled Y, which corresponds to the production rule $Z \to XY$.

If a CFG is obtained from any other CFG by a permutation $\pi : \Sigma \cup V \to \Sigma \cup V$, they are identical to each other because the string derived from one is transformed to that from the other by the renaming. For example, $P = \{Z \to XY, Y \to ab, X \to aa\}$ and $P' = \{X \to YZ, Z \to ab, Y \to aa\}$ are identical each other. On the other hand, they are clearly different from $P'' = \{Z \to aY, Y \to$

$bX, X \to aa\}$ because their depths are different. Thus, we assume the following canonical form of CFG called *straight line program* (SLP).

Definition 1. *(Karpinsk-Rytter-Shinohara [18]) An SLP is a grammar compression over $\Sigma \cup V$ whose production rules are formed by either $X_i \to a$ or $X_k \to X_i X_j$, where $a \in \Sigma$ and $1 \leq i, j < k \leq |V|$.*

2.2 Phrase/reverse dictionary

For a set P of production rules, a *phrase dictionary* D is a data structure for directly accessing the phrase $X_i X_j$ for any $X_k \in V$ if $X_k \to X_i X_j \in P$. Regarding a triple (k, i, j) of positive integers as $X_k \to X_i X_j$, we can store the phrase dictionary consisting of n variables in an integer array $D[1, 2n]$, where $D[2k-1] = D[2k] = 0$ if k belongs to an alphabet i.e., $1 \leq k \leq |\Sigma|$. X_i and X_j are accessible as $D[2k-1]$ and $D[2k]$ by indices $2k-1$ and $2k$ for X_k, respectively. A plain representation of D using fixed-length codes requires $2n \log n$ bits of space to store n production rules.

Reverse dictionary D^{-1} is a data structure for directly accessing the variable X_k given $X_i X_j$ for a production rule $X_k \to X_i X_j \in P$. Thus, $D^{-1}(X_i X_j)$ returns X_k if $X_k \to X_i X_j \in P$. A hash table is a representative data structure for D^{-1} enabling $O(1)$ time access and achieving $O(n \log n)$ bits of space.

2.3 Rank/select dictionary

We present a phrase dictionary based on the *rank/select dictionary*, a data structure for a bit string B [13] supporting the following queries: $\mathrm{rank}_c(B, i)$ returns the number of occurrences of $c \in \{0, 1\}$ in $B[1, i]$ and $\mathrm{select}_c(B, i)$ returns the position of the i-th occurrence of $c \in \{0, 1\}$ in B. For example, if $B = 10110100111$ is given, then $\mathrm{rank}_1(S, 7) = 4$ because the number of 1s in $B[1, 7]$ is 4, and $\mathrm{select}_1(S, 5) = 9$ because the position of the fifth 1 in B is 9. Although naive approaches require the $O(|B|)$ time to compute a rank, several data structures with only the $|B| + o(|B|)$ bit storage to achieve $O(1)$ time [26,27] have been presented. Most methods compute a select query by a binary search on a bit string B in $O(\log |B|)$ time. A data structure for computing the select query in $O(1)$ time has also been presented [28].

2.4 Wavelet tree

A WT is a data structure for a string $S \in \Sigma^*$, and it can be used to compute the rank and select queries on a string S over an ordinal alphabet in $O(\log \sigma)$ time and $n \log \sigma(1 + o(1))$ bits [12]. Data structures supporting the rank and select queries in $O(\log \log \sigma)$ time with the same space have been proposed [10,3]. WT also supports access(S, i) which returns $S[i]$ in $O(\log \sigma)$ time. Recently, WT has been extended to support various operations on strings [25].

A WT for a sequence S over $\Sigma = \{1, ..., \sigma\}$ is a binary tree that can be, recursively, presented over a sub-alphabet range $[a, b] \subseteq [1, \sigma]$. Let S_v be a sequence

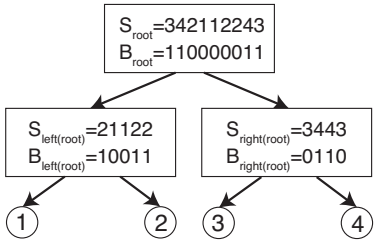

Fig. 1. Example of wavelet tree for a sequence $S = 342112243$ over an alphabet $\{1, 2, 3, 4\}$.

represented in a node v, and let $left(v)$ and $right(v)$ be left and right children of node v, respectively. The root v_{root} represents $S_{root} = S$ over the alphabet range $[1, \sigma]$. At each node v, S_v is split into two subsequences $S_{left(v)}$ consisting of the sub-alphabet range $[a, \lfloor \frac{(a+b)}{2} \rfloor]$ for $left(v)$ and $S_{right(v)}$ consisting of the sub-alphabet range $[\lfloor \frac{(a+b)}{2} \rfloor + 1, b]$ for $right(v)$ where $S_{left(v)}$ and $S_{right(v)}$ keep the order of elements in S_v. The splitting process repeats until $a = b$. Each node v in the binary tree contains a rank/select dictionary on a bit string B_v. Bit $B_v[k]$ indicates whether $S_v[k]$ should be moved to $left(v)$ or $right(v)$. If $B_v[k] = 0$, $S_{left(v)}$ contains $S_v[k]$. If $B_v[k] = 1$, $S_{right(v)}$ inherits $S_v[k]$. Formally, $B_v[k]$ with an alphabet range $[a, b]$ is defined as:

$$B_v[k] = \begin{cases} 1 \text{ if } S_v[k] > \lfloor (a+b)/2 \rfloor \\ 0 \text{ if } S_v[k] \leq \lfloor (a+b)/2 \rfloor \end{cases}.$$

An example of a WT is shown in Figure 1. In this example, since $S_{root}[2] = 4$ belongs to the higher half $[3, 4]$ of an alphabet range $[1, 4]$ represented in the root; therefore, it is the second element of S_{root} that must go to the right child of the root, $B_{root}[2] = 1$ and $S_{right(root)}[2] = S_{root}[2] = 4$.

3 Succinct SLP

3.1 Information-theoretic lower bound

In this section, we present a tight lower bound to represent SLPs having a set of production rules P consisting of $n = |\Sigma \cup V|$ symbols. Each production rule $Z \to XY \in P$ is considered as two directed edges (Z, X) and (Z, Y), the SLP can be seen as a directed acyclic graph (DAG) with a single source and $|\Sigma|$ sinks. Here, we consider (Z, X) as the left edge and (Z, Y) as the right edge. In addition, P can be considered as a DAG with the single source and with a single sink by introducing a super-sink s and drawing directed left and right edges from any sink to s (Figure 2). Let $\mathcal{DAG}(n)$ be the set of all possible Gs with n nodes and $\mathcal{DAG} = \bigcup_{n \to \infty} \mathcal{DAG}(n)$. Since two SLPs are identical if an SLP can be converted to the other SLP by a permutation $\pi : \Sigma \cup V \to \Sigma \cup V$, the number of different SLPs is $|\mathcal{DAG}(n)|$. Any internal node of $G \in \mathcal{DAG}(n)$

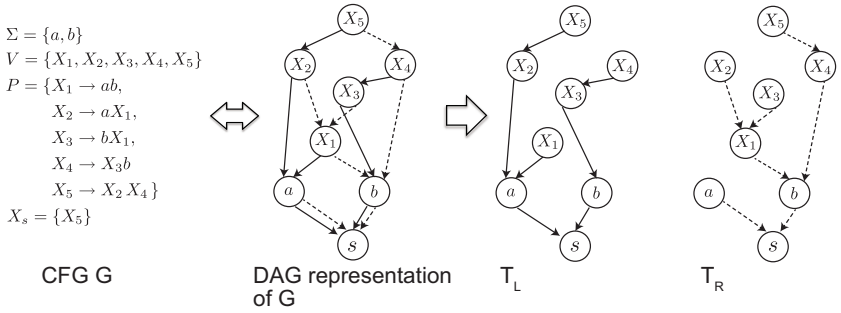

Fig. 2. Example of DAG representation of an SLP and its spanning tree decomposition. An SLP is represented by a DAG G. G is decomposed into the left tree T_L and right tree T_R.

has exactly two (left/right) edges. Thus, the following fact remarked in [22] is true.

***Fact* 1** *An in-branching spanning tree is an ordered tree such that the out-degree of any node except the root is exactly one. For any in-branching spanning tree of G, the graph consisting of the remaining edges and their adjacent vertices is also an in-branching spanning tree of G.*

The in-branching spanning tree consisting of the left edges (respectively the right edges) and their adjacent vertices is called the *left tree* T_L (respectively *right tree* T_R) of G. Note that the source in G is a leaf of both T_L and T_R, and the super-sink of G is the root of both T_L and T_R. We shall call the operation of decomposing a DAG G into two spanning trees T_L and T_R *spanning tree decomposition*. In Figure 2, the source x_5 in G is a leaf of both T_L and T_R, and the super-sink s in G is the root of both T_L and T_R.

Any ordered tree is an elements in $\mathcal{T} = \bigcup_{n\to\infty} \mathcal{T}_n$ where $\mathcal{T}_n$ is the set of all possible ordered trees with n nodes. As shown in [2,34], there exists an enumeration tree for $\mathcal{T}$ such that any $T \in \mathcal{T}$ appears exactly once. The enumeration tree is defined by the *rightmost expansion*, i.e., in this enumeration tree, a node $T' \in \mathcal{T}_{n+1}$, which is a child of $T \in \mathcal{T}_n$, is obtained by adding a rightmost node to T. In our problem, an ordered tree $T \in \mathcal{T}_{n+1}$ is identical to a left tree T_L with $n+1$ nodes for $n = |\Sigma \cup V|$ symbols.

Let $G \oplus (u, v)$ be the DAG obtained by adding the edge (u, v) to a DAG G. If necessary, we write $G \oplus (u, v)_L$ to indicate that (u, v) is added as a left edge. For a set E of edges, the DAG $G \oplus E$ is defined analogously. The DAG $G \oplus E$ is defined as adding all the edges $(u, v) \in E$ to G. The DAG $G \ominus E$ is also defined as deleting all the edges $(u, v) \in E$ from G.

Theorem 1. *The information-theoretic lower bound on the minimum number of bits needed to represent an SLP with n symbols is $2n + \log n! + o(n)$.*

Proof. Let $\mathcal{S}(n)$ be the set of all possible DAGs with n nodes and a single source/sink such that any internal node has exactly two children. This $\mathcal{S}(n)$ is a

super set of $\mathcal{DAG}(n)$ because the in-degree of the sink of any DAG in $\mathcal{DAG}(n)$ must be exactly 2σ, whereas $\mathcal{S}(n)$ does not have such a restriction. By the definition, $|\mathcal{S}(n)|/n^\sigma \leq |\mathcal{DAG}(n)| \leq |\mathcal{S}(n)|$ holds.

Let $\mathcal{S}(n,T) = \{G \in \mathcal{S}(n) \mid G = T \oplus T_R,\ T_R \in \mathcal{T}_n\}$. We show $|\mathcal{S}(n,T)| = (n-1)!$ for each $T \in \mathcal{T}_n$ by induction on $n \geq 1$. Since the base case $n = 1$ is clear, we assume that the induction hypothesis is true for some $n \geq 1$.

Let T'_L be the rightmost expansion of T_L such that the rightmost node u is added as the rightmost child of node v in T_L, and let $G' \in \mathcal{S}(n+1, T'_L)$ with a left tree T'_L. By the induction hypothesis, the number of $G \in \mathcal{S}(n, T_L)$ is $(n-1)!$ and T_L is embedded into G as the left tree. Then, G' is constructed by adding the left edge (u,v) and a right edge (u,x) for a node x in T_L.

Let s be the source of G. For $v = s$, each $G' = G \oplus (u,v)_L \oplus (u,x)_R \in \mathcal{S}(n+1, T'_L)$ is admissible, and the number of them is clearly $n|\mathcal{S}(n, T_L)| = n!$. For $v \neq s$, if $x = s$, $G' = G \oplus (u,v)_L \oplus (u,x)_R \in \mathcal{S}(n+1, T'_L)$ is admissible.

Otherwise, there exists the lowest common ancestor y of s and x on T_R with $G = T_L \oplus T_R$. Let z be the unique child of y and and let $p(s,z)$ be the path of T_R from z' to z, where possibly $s = z$. If the in-dgree of any node in $p(s,z)$ is at most one in G, we generate $G' = G \oplus (u,v)_L \oplus (u,x)_R \ominus (z,y)_R \oplus (z,u)_R$. If G' contains a cycle, it must contain the edge (z,u). However, there is no such a path because of the condition of $p(s,z)$. Thus, G' is an admissible DAG in $\mathcal{S}(n+1, T'_L)$. Conversely, if some node in $p(s,z)$ is more than two in G, let z' be the nearest one from s. Analogously, $G' = G \oplus (u,v)_L \oplus (u,x)_R \ominus (z',z)_R \oplus (z',u)_R$ is an admissible DAG in $\mathcal{S}(n+1, T'_L)$.

In all the cases, the number of such G's is also $n!$ because no edge is changed in T_L and the pair (T'_L, T'_R) containing the edge $(u,x)_R$ is unique for any fixed T'_L. Thus, $|\mathcal{S}(n+1,T)| = n!$ is true for each $T \in \mathcal{T}_{n+1}$.

This result derives $|\mathcal{S}(n)| = C_n(n-1)!$ where $C_n = \frac{1}{n+1}\binom{2n}{n} \simeq 2^{2n}n^{-3/2}$ is the number of ordered trees with $n+1$ nodes. Combining this with $|\mathcal{S}(n)|/n^\sigma \leq |\mathcal{G}(n)| \leq |\mathcal{S}(n)|$ as well, we get the result that the information-theoretic minimum bits needed to represent $G \in \mathcal{DAG}(n)$ is at least $2n + \log n! + o(n)$. □

3.2 An optimal SLP representation

We present an optimal reresentation of an SLP as an improvement of the data structure recently presented in [32]. We apply the spanning tree decomposition to the DAG G of a given SLP, and obtain the DAG $T_L \oplus T_R (= G)$. We rename the variables in T_L by breadth-first order and also rename variables in T_R according to the T_L. Let G' be the resulting DAG from G. Then, for the array representation $D[1,2n]$ of G', we obtain the condition $D[1] \leq D[3] \leq \ldots \leq D[2n-1]$. Since this monotonic sequence is encoded by $2n + o(n)$ bits, D is represented by $2n + n\log n + o(n)$ bits supporting $access(D,k)$ $(1 \leq k \leq 2n)$ in $O(1)$ time. We focus on the remaining sequence of length n, i.e., $D[2], D[4], \ldots, D[2n]$. For simplicity, we write D instead of $[D[2], D[4], \ldots, D[2n]]$.

Let $\mathcal{S} = \{s_1, \ldots, s_\rho\}$ be a disjoint set of subsequences of $[1,n]$ such that any $i \in \{1,2,...,n\}$ is contained in some s_k and any s_i, s_j $(i \neq j)$ are disjoint. Such an $\mathcal{S}$ is called a decomposition of D. A sequence $D[s_{k_1}], \ldots, D[s_{k_p}]$

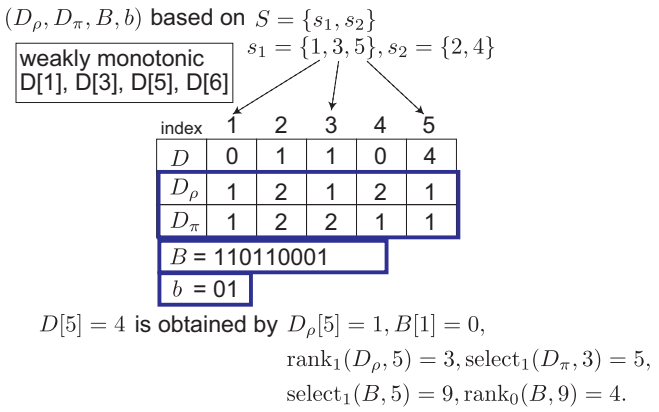

Fig. 3. Encoded phrase dictionary: D indicates the remaining sequence $D[2], D[4], \ldots, D[2n]$. D is encoded by $(D_\rho, D_\pi, \mathbf{B}, \mathbf{b})$ based on a monotonic decomposition $\mathcal{S}$ of D, i.e., each $s \in \mathcal{S}$ indicates a weakly monotonic subsequence in D; D_ρ is the sequence of i indicating the membership for some $s_i \in \mathcal{S}$, D_π is a permutation of D_ρ with respect to the corresponding value in D, $\mathbf{B}$ is a binary encoding of the sorted D in increasing order. We show only the case that $D[i]$ is a member of an increasing $s \in \mathcal{S}$, but the other case is similarly computed by $\mathbf{b}$.

is weakly monotonic if it is *increasing*, i.e., $D[s_{k_1}] \leq \ldots \leq D[s_{k_p}]$ or *decreasing*, i.e., $D[s_{k_1}] \geq \ldots \geq D[s_{k_p}]$. In addition, $\mathcal{S}$ is called *monotonic* if the sequence $D[s_{k_1}], \ldots, D[s_{k_p}]$ is weakly monotonic for any $s_k = [s_{k_1}, \ldots, s_{k_p}] \in \mathcal{S}$.

Theorem 2. *Any SLP with n symbols can be represented using $2n \log \rho(1+o(1))$ bits for $\rho \leq 2\sqrt{n}$, while supporting $O(\log \log \rho)$ access time.*

Proof. It is sufficient to prove that any D of length n can be represented using $2n \log \rho + o(n)$ bits for some $\rho \leq 2\sqrt{n}$. By the result in [33], we can construct a monotonic decomposition $\mathcal{S}$ of D such that $\rho = |\mathcal{S}| \leq 2\sqrt{n}$.

We represent the sequence D as a four-tuple $(D_\rho, D_\pi, \mathbf{B}, \mathbf{b})$ using $\mathcal{S}$. For each $1 \leq p \leq n$, $D_\rho[p] = k$ iff p is a member of $s_k \in \mathcal{S}$ for some $1 \leq k \leq \rho$. Let $(D[1], D_\rho[1]), \ldots, (D[n], D_\rho[n])$ be the sequence of pairs $(D[p], D_\rho[p])$ $(1 \leq p \leq n)$. We sort these pairs with respect to the keys $D[p]$ $(1 \leq p \leq n)$ and obtain the sorted sequence $(D[\ell_1], D_\rho[\ell_1]), \ldots, (D[\ell_n], D_\rho[\ell_n])$. We define D_π as the permutation $D_\rho[\ell_1] \cdots D_\rho[\ell_n]$.

$\mathbf{B} \in \{0, 1\}^*$ is defined as the bit string

$$\mathbf{B} = 0^{D[\ell_1]} 10^{D[\ell_2] - D[\ell_1]} \cdots 10^{D[\ell_n] - D[\ell_{n-1}]} 1.$$

Finally, $\mathbf{b}[k] = 0$ if $s_k \in \mathcal{S}$ is increasing and $\mathbf{b}[k] = 1$ otherwise for $1 \leq k \leq \rho$. D and D_ρ are represented by WTs, respectively, and $\mathbf{B}$ is a rank/select dictionary.

We recover $D[p]$ using $(D_\rho, D_\pi, \mathbf{B}, \mathbf{b})$. When $D_\rho[p] = k$ and $\mathbf{b}[k] = 0$, i.e., $D[p]$ is included in the k-th monotonic subsequence $s_k \in \mathcal{S}$ that is increasing,

we obtain

$$D[p] = \mathrm{rank}_0(\mathbf{B}, \mathrm{select}_1(\mathbf{B}, \ell))$$

by $\ell = \mathrm{select}_k(D_\pi, \mathrm{rank}_k(D_\rho, p))$. When $D_\rho[p] = k$ and $\mathbf{b}[k] = 1$, we can similarly obtain $D[p]$ replacing ℓ by $r = \mathrm{select}_k(D_\pi, (\mathrm{rank}_k(D_\rho, n) + 1 - \mathrm{rank}_k(D_\rho, p)))$.

The total size of the data structure formed by $(D_\rho, D_\pi, \mathbf{B}, \mathbf{b})$ is at most $2n \log \rho(1 + o(1))$ bits. The rank/select/access operations of the WT for a static sequence over $\rho \leq 2\sqrt{n}$ symbols can be improved to achieve $O(\log \log \rho)$ time for each query [3,10]. □

In Figure 3, for the sequence $(0,1),(1,2),(1,1),(0,2),(4,1)$ of pairs $(D[p], D_\rho[p])$ $(1 \leq p \leq 5)$, the sorted sequence is $(0,1),(0,2),(1,2),(1,1),(4,1)$. Thus, D_π is 12211. $\mathbf{B} = 0^0 10^{(0-0)} 10^{(1-0)} 10^{(1-1)} 10^{(4-1)} 1 = 110110001$. $b[1] = 0$ because s_1 is increasing, and $b[2] = 1$ because s_2 is decreasing.

4 Data Structure for Reverse Dictionary

In this section, we present a data structure for simulating the naming function H defined as follows. For a phrase dictionary D with n symbols,

$$H(X_i X_j) = \begin{cases} D^{-1}(X_i X_j), & \text{if } D[k] = X_i X_j \text{ for some } 1 \leq k \leq n, \\ X_{n+1}, & \text{otherwise.} \end{cases}$$

For a sufficiently large V, we set a total order on $(\Sigma \cup V)^2 = \{XY \mid X, Y \in \Sigma \cup V\}$, i.e., the lexicographical order of the n^2 digrams. This order is represented by the range $[1, n^2]$. Then, we recursively define WT T_D for a phrase dictionary D partitioning $[1, n^2]$. On the root node, the initial range $[1, n^2]$ is partitioned into two parts: a left range $L[1, \lfloor (1+n^2) \rfloor / 2]$ and a right range $R[\lfloor (1+n^2) \rfloor / 2 + 1, n^2]$. The root is the bit string $\mathbf{B}$ such that $\mathbf{B}[i] = 0$ if $D[i] \in L$ and $\mathbf{B}[i] = 1$ if $D[i] \in R$. By this, the sequence of digrams, D, is decomposed into two subsequences D_L and D_R; they are projected on the roots of the left and right subtrees, respectively. Each sub-range is recursively partitioned and the subsequence of D on a node is further decomposed with respect to the partitioning on the node. This process is repeated until the length of any sub-range is one. Let $\mathbf{B}_i$ be the bit string assigned to the i-th node of T_D in the breadth-first traversal. In Figure 4, we show an example of such a data structure for a phrase dictionary D.

Theorem 3. *The naming function for phrase dictionary D over $n = |\Sigma \cup V|$ symbols can be computed by the proposed data structure D_T in $O(\log n)$ time for any digram. Moreover, when a digram does not exist in the current D, D_T can be updated in the same time and the space is at most $2n \log n(1 + o(1))$ bits.*

Proof. D_T is regarded as a WT for a string S of length n such that any symbol is represented in $2 \log n$ bits. Thus, $H(XY)$ is obtained by $\mathrm{select}_{XY}(S, 1)$. The query time is bounded by the number of rank and select operations for bit strings performed until the operation flow returns to the root. Since the total range is

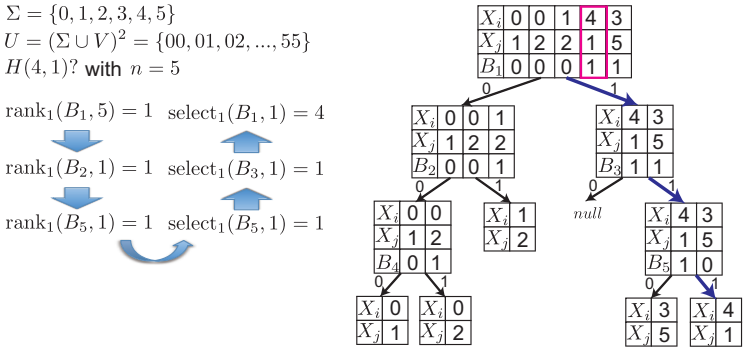

Fig. 4. WT for reverse dictionary: The bit string $\mathbf{B}_i$ is assigned to the i-th node in breadth-first order. For each internal node i, we can move to the left child by rank_0 and to the right child by rank_1 on $\mathbf{B}_i$. The upward traversal is simulated by select_0 and select_1 as shown. The leaf for an existing digram is represented by 1 and *null* is represented by 0, whereas these bits are omitted in this figure.

$[1, n^2]$, i.e., the height of T_D is at most $2 \log n$, the query time and the size are derived. When XY does not exist in D, let $i_1, i_2, \ldots, i_k$ be the sequence of traversed nodes from the root i_1 to a leaf i_k and let $\mathbf{B}_{i_j}$ be the bit string on i_j. Given an access/rank/select dictionary for $\mathbf{B}_{i_j}$, we can update it for $\mathbf{B}_{i_j} b$ and $b \in \{0, 1\}$ in $O(1)$ time. Therefore, the update time of T_D for any digram is $O(k) = O(\log n)$. □

5 Discussion

We have investigated three problems related to the construction of an SLP: the information-theoretic lower bound for representing the phrase dictionary D, an optimal representation of a directly addressable D, and a dynamic data structure for D^{-1}. Here, we consider the results of this study from the viewpoint of open questions.

For the first problem, we approximately estimated the size of a set of SLPs with n symbols, which is almost equal to the exact set. This problem, however, has several variants, e.g., the set of SLPs with n symbols deriving the same string, which is quite difficult to estimate owing to the NP-hardness of the smallest CFG problem. There is another variant obtained by a restriction: Any two different variables do not derive the same digram, i.e., $Z \to XY$ and $Z' \to XY$ do not exist simultaneously for $Z \neq Z'$. Although such variables are not prohibited in the definition of SLP, they should be removed for space efficiency. On the other hand, even if we assume this restriction, the information-theoretic lower bound is never smaller than $\log n!$ bits because, given a directed chain of length n as T_L, we can easily construct $(n-1)!$ admissible DAGs.

For the second problem, we proposed almost optimal encoding of SLPs. From the standpoint of massive data compression, one drawback of the proposed encoding is that the whole phrase dictionary must be stored in memory beforehand. Since symbols must be sorted, we need a dynamic data structure to allow the insertion of symbols in an array, e.g., [15]. Such data structures, however, require $O(n \log n)$ bits of space.

For the last problem, the query time and update time of proposed data structure are both $O(\log n)$. This cost is considerable and it is difficult to improve it to $O(\log \log n)$ because D is not static. When focusing on the characteristics of SLPs, we can improve the query time probabilistically; since any symbol X appears in D at least once and $|D| = 2n$, the average of frequency of X is at most two. Thus, using an additional array of size $n \log n$ bits, we can check $H(XY)$ in $O(1)$ time with probability at least $1/2$. However, improving this probability is not easy. For this problem, achieving $O(1)$ amortized query time is also an interesting challenge.

References

1. A. Apostolico and S. Lonardi. Off-line Compression by Greedy Textual Substitution. *Proceedings of the IEEE*, 88:1733–1744, 2000.
2. T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient Substructure Discovery from Large Semi-structured Data. In *SDM*, pages 158–174, 2002.
3. J. Barbay, T. Gagie, G. Navarro, and Y. Nekrich. Alphabet Partitioning for Compressed Rank/Select and Applications. In *ISAAC*, volume 2, pages 315–326, 2010.
4. J. Barbay and G. Navarro. Compressed Representations of Permutations, and Applications. In *STACS*, pages 111–122, 2009.
5. F.C. Botelho, R. Pagh, and N. Ziviani. Simple and Space-Efficient Minimal Perfect Hash Functions. In *WADS*, pages 139–150, 2007.
6. M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inform. Theory*, 51:2554–2576, 2005.
7. F. Claude and G. Navarro. Self-Indexed Grammar-Based Compression. *Fundam. Inform.*, 111:313–337, 2011.
8. P. Ferragina and R. Venturini. A simple storage scheme for strings achieving entropy bounds. *Theor. Comput. Sci.*, 372:115–121, 2007.
9. F.V. Fomin, D. Kratsch, and J.-C. Novelli. Approximating minimum cocolorings. *Inf. Process. Lett.*, 84:285–290, 2002.
10. A. Golynski, J.I. Munro, and S.S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *SODA*, pages 368–373, 2006.
11. R. González and G. Navarro. Statistical Encoding of Succinct Data Structures. In *CPM*, pages 294–305, 2006.
12. R. Grossi, A. Gupta, and J.S. Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.
13. G. Jacobson. Space-efficient Static Trees and Graphs. In *FOCS*, pages 549–554, 1989.
14. J. Jansson, K. Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees with applications. *J. Comput. Syst. Sci.*, 78:619–631, 2012.

15. J. Jansson, K. Sadakane, and W.K. Sung. CRAM: Compressed Random Access Memory. In *ICALP*, volume 1, pages 510–521, 2012.
16. R.M. Karp, R.E. Miller, and A.L. Rosenberg. Rapid Identification of Repeated Patterns in Strings, Trees and Arrays. In *STOC*, pages 125–136, 1972.
17. R.M. Karp and M.O. Rabin. Efficient Randomized Pattern-Matching Algorithms. *IBM Journal of Research and Development*, 31:249–260, 1987.
18. M. Karpinski, W. Rytter, and A. Shinohara. An Efficient Pattern-Matching Algorithm for Strings with Short Descriptions. *Nordic J. Comp.*, 4:172–186, 1997.
19. N.J. Larsson and A. Moffat. Offline Dictionary-Based Compression. In *DCC*, pages 296–305, 1999.
20. E. Lehman. *Approximation Algorithms for Grammar-Based Compression*. PhD thesis, MIT, 2002.
21. E. Lehman and A. Shelat. Approximation algorithms for grammar-based compression. In *SODA*, pages 205–212, 2002.
22. S. Maruyama, M. Nakahara, N. Kishiue, and H. Sakamoto. ESP-Index: A Compressed Index Based on Edit-Sensitive Parsing. In *SPIRE*, pages 398–409, 2011.
23. S. Maruyama, M. Nakahara, N. Kishiue, and H. Sakamoto. ESP-Index: A Compressed Index Based on Edit-Sensitive Parsing. *J. Discrete Algorithms*, 18:100–112, 2013.
24. S. Maruyama, H. Sakamoto, and M. Takeda. An Online Algorithm for Lightweight Grammar-Based Compression. *Algorithms*, 5:213–235, 2012.
25. G. Navarro. Wavelet Trees for All. In *CPM*, pages 2–26, 2012.
26. G. Navarro and E. Providel. Fast, Small, Simple Rank/Select on Bitmaps. In *SEA*, pages 295–306, 2012.
27. D. Okanohara and K. Sadakane. Practical Entropy-Compressed Rank/Select Dictionary. In *ALENEX*, 2007.
28. R. Raman, V. Raman, and S.S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *SODA*, pages 233–242, 2002.
29. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302:211–222, 2003.
30. K. Sadakane and R. Grossi. Squeezing succinct data structures into entropy bounds. In *SODA*, pages 1230–1239, 2006.
31. H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *J. Discrete Algorithms*, 3:416–430, 2005.
32. Y. Takabatake, Y. Tabei, and H. Sakamoto. Variable-Length Codes for Space-Efficient Grammar-Based Compression. In *SPIRE*, pages 398–410, 2012.
33. R.B. Yehuda and S. Fogel. Partitioning a Sequence into Few Monotone Subsequences. *Acta Inf.*, 35:421–440, 1998.
34. M.J. Zaki. Efficiently mining frequent trees in a forest. In *KDD*, pages 71–80, 2002.
35. J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inform. Theory*, 23:337–343, 1977.
36. J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inform. Theory*, 24:530–536, 1978.