JMLR: Workshop and Conference Proceedings 34:3–20, 2014 Proceedings of the 12th ICGI

# Grammar Compression: Grammatical Inference by Compression and Its Application to Real Data

**Hiroshi Sakamoto** HIROSHI@AI.KYUTECH.AC.JP
*Kyushu Institute of Technology, Japan*

**Editor:** Alexander Clark, Makoto Kanazawa and Ryo Yoshinaka

## Abstract

A grammatical inference algorithm tries to find as a small grammar as possible representing a potentially infinite sequence of strings. Here, let us consider a simple restriction: the input is a finite sequence or it might be a singleton set. Then the restricted problem is called the *grammar compression* to find the smallest CFG generating just the input. In the last decade many researchers have tackled this problem because of its scalable applications, e.g., expansion of data storage capacity, speeding-up information retrieval, DNA sequencing, frequent pattern mining, and similarity search. We would review the history of grammar compression and its wide applications together with an important future work. The study of grammar compression has begun with the bad news: the smallest CFG problem is NP-hard. Hence, the first question is: Can we get a near-optimal solution in a polynomial time? (Is there a reasonable approximation algorithm?) And the next question is: Can we minimize the costs of time and space? (Does a linear time algorithm exist within an optimal working space?) The recent results produced by the research community answer affirmatively the questions. We introduce several important results and typical applications to a huge text collection. On the other hand, the shrinkage of the advantage of grammar compression is caused by the data explosion, since there is no working space for storing the whole data supplied from data stream. The last question is: How can we handle the stream data? For this question, we propose the framework of *stream grammar compression* for the next generation and its attractive application to fast data transmission.

## 1. Introduction

The aim of an inference or learning algorithm is to find a compact representation of a potentially infinite sequence of string $w_i$. Depending on the class the input belongs to, this problem becomes harder and harder to implement the algorithm, whereas the real world data we can observe is always finite. One idea is to give up the prediction of future, in other words, to consider a currently optimal solution for a finite input $w_1, w_2, \dots, w_n$. When regarding the delimiter "," as a special character #, this problem is transformed to the one to find a minimum grammar generating the single string $w = w_1\#w_2\#\cdots\#w_n$. If the right linear grammar (or DFA) is enough for your needs, it is easy to get a smallest DFA for $w$. However you might need more compact representation of $w$ due to the scale of data you have, for instance, over a few hundred gigabytes of genome sequences.

As far as I know, polynomial time inferring CFG from infinite sequence is probably impossible. For the finite case $w$, this problem is easy if any CFG is admissible (e.g., consider the trivial one consisting of $S \to w$ only). Then, we are interested in the smallest

© 2014 H. Sakamoto.

CFG generating just $w$. Unfortunately, this question was negatively solved by Lehman and Shelat (2002) in 2002. They showed a reduction from the independent set to the minimum CFG, that is, the NP-hardness and the non-approximability within a constant ratio unless P = NP. On one hand, Charikar et al. (2005) showed a positive result: the smallest CFG is approximable within a logarithmic ratio, a milestone in the grammar compression. Besides, other approximation algorithms (Rytter, 2003; Sakamoto, 2005) achieving the same approximation ratio were simultaneously proposed.

Because these algorithms run in linear time in the size of input, we got the best lower bound of the time complexity. Then the next challenge is consisting of two tasks: an optimal encoding of the resulting CFG and the improvement of the space complexity. Here we should note the importance of these tasks, since they are necessary for practical application of the grammar compression.

First, we consider the importance of the encoding CFG. Any CFG $G$ can be transformed into another $G'$ in Chomsky normal form with $|G'| \leq 2|G|$. Therefore, we can assume any CFG is of the form throughout this paper. Given a set $P$ of $n$ production rules, the CFG is represented by an integer array $A[1, 2n]$ such that $A[2k-1, 2k]$ stores the integers $i, j$ iff $X_k \to X_i X_j \in P$. The fixed-length encoding of $A[1, 2n]$ requires $2n \lg n$[1] bits. A merit of array representation is that we can randomly access to any production rule in $O(1)$ time. Instead, we must pay a considerable space for it. For example, let us examine the influence for a real text. Given the text "Alice's Adventures in Wonderland" by Lewis Carroll of size 152KB to a standard grammar compression algorithm, it outputs the array $A[1, 2n]$ with $n = 45,243$. If taking 32-bit integer for the fixed-length code, the size of $A$ is in fact 177KB, that is, the algorithm compresses the input to a larger output!

Huffman encoding is an optimal solution for the absurdity, and many grammar compression and related algorithms (e.g., Ziv and Lempel, 1977, 1978; Welch, 1984; Larsson and Moffat, 2000) actually adopt it. However, since Huffman encode does not support the random access on the string, the range of application is extremely restricted, for example, only the partial decoding cannot be performed. This big problem has been overcome using *succinct data structures* and related works (Jacobson, 1989; Munro and Raman, 2001; Raman et al., 2002; Munro et al., 2003; Grossi et al., 2003; Benoit et al., 2005; Sadakane and Navarro, 2010; Lu and Yeh, 2008). For the class $C = \{x_1, x_2, \ldots, x_n\}$ of $n$ objects, $\lceil \lg n \rceil$ is the minimum bits to represent any $x_i \in C$. If a representation method requires $n + o(n)$ bits for any $x_i \in C$, the representation is called *succinct*. Using such techniques, several succinct CFG representations have been proposed (Sakamoto et al., 2009; Maruyama et al., 2012; Takabatake et al., 2012) while supporting the traverse of the derivation tree. Moreover, the information-theoretic lower bound was also proved (Tabei et al., 2013b).

Second, we mention the other important task: reducing space complexity. There are many algorithms having good capability for compression, but there is a trade-off between the output size and the working space. For example, Repair (Larsson and Moffat, 2000) is one of the best-performed algorithm, however, the required working memory is over 10 times greater than the input size.

Against this drawback, many researchers have challenged the space-saving grammar compression, and proposed efficient algorithms as well as their applications to information

1. lg stands for $\log_2$.

retrieval and data mining, where ”space-saving” means that the space is bounded by the output size.

An important application is the *self-index* (Ferragina et al., 2007; Sadakane, 2003), an index of text $S$ supporting counting/locating of pattern $P$ for $S[i,j] = P$ and partial decompression $S[i,j]$ without explicit $S$. Whereas almost self-indexes were based on the Burrows Wheeler Transform (Burrows and Wheeler, 1994), the size of required memory is proportional to the size of input. Because of this bottleneck, grammar compression have attracted the attentions and several algorithms have been proposed (Claude and Navarro, 2011; Kreft and Navarro, 2011). Recently the required memory is significantly improved (Maruyama et al., 2013a; Takabatake et al., 2014a). Such algorithms can be expanded to the frequent pattern mining (Nakahara et al., 2013) and the edit edit distance problem (Takabatake et al., 2014b), which are inspired by the pioneering study indicating the relation between the compression and the text similarity (Li et al., 2004; Cilibrasi and Vitanyi, 2005).

In the above, we have introduced the grammar compression for *static data*. On the other hand, the stream data is another interesting application of grammar compression, where the input sequence is given one by one and the algorithm must output for the latest input. Recently, a *fully-online* algorithm for this problem have been proposed (Maruyama et al., 2013b) and it was applied to the online computation of an extended the edit distance (Takabatake et al., 2014b). The study of online grammar compression is developing and the online construction of self-index by grammar compression is one of the most important open problem.

## 2. Grammar Compression

### 2.1. The problem and its hardness

$|C|$ denotes the cardinality of the set $C$. $\Sigma$ is a finite set of symbols and we assume $|\Sigma|$ is a constant. Let $\mathcal{X}$ be 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 $\Sigma$ is $\Sigma^*$. 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 $G = (\Sigma, V, P, X_s)$ 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. The set of strings in $\Sigma^*$ derived from $X_s$ by $G$ is denoted by $L(G)$. A CFG $G$ is called *admissible* if exactly one $X \to \alpha \in P$ exists and $|L(G)| = 1$. An admissible $G$ deriving $S$ is called a *grammar compression* of $S$ for any $X \in V$. The size of $G$, $|G|$, is the sum of $|\alpha|$ for all $X \to \alpha \in P$.

**Problem 1** Minimum Grammar Compression
Instance: *A string $w \in \Sigma^*$.*
Solution: *A smallest grammar compression $G$ of $w$.*

In 2002 the NP-hardness has been proved as well as there is no polynomial time approximation algorithm with ratio less than 5761/5760 unless P = NP (Lehman and Shelat, 2002). Thus, the next our interest is whether there is an approximation algorithm with a reasonable ratio. Of course, $O(N)$-approximation is trivial for $N = |w|$. In the following several years, three $O(\lg N)$-algorithms (Rytter, 2003; Charikar et al., 2005; Sakamoto, 2005) were independently proposed. The simplest algorithm is based on a canonical form of CFG, called straight-line program, and related techniques.

## 2.2. Straight-line program

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 (Ziv and Lempel, 1978), REPAIR (Larsson and Moffat, 2000) and LCA (Maruyama et al., 2012), and indexes e.g. SLP (Claude and Navarro, 2011) and ESP (Maruyama et al., 2011).

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 $Y$, corresponding to the $Z \to XY \in P$.

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. Thus, we assume the following canonical form of CFG.

**Definition 1** **(Karpinski et al., 1997)** *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* $1 \leq i, j < k \leq |V|$.

Without loss of generality we can assume a smallest CFG is given as an SLP since the required ratio is $O(\lg N)$. One of the first $O(\lg N)$-approximation ratio was proved by the relation between the size of $G$ and the length of *LZ-factorization*, a decomposition of $w$ into a sequence of its substrings. Next, we focus on this analysis.

## 2.3. Approximation algorithm

It is known that there is an important relation between a deterministic CFG and a simple decomposition of $w$. The *LZ-factorization* $LZ(w)$ *of* $w$ is the sequence $f_1 \cdots\cdots f_k$, where $f_1 = w[1]$, and for each $1 < \ell \leq k$, $f_\ell$ is the longest prefix of the suffix $w[|f_1 \cdots f_{\ell-1}| + 1, |w|]$ that appears in $f_1 \cdots f_{\ell-1}$. The size $|LZ(w)|$ of $LZ(w)$ is the number of its factors. For example, if $w = ababababaabab$, $LZ(w) = a, b, ab, aba, abab$ and $|LZ(w)| = 5$. For this factorization of $w$, the following relation between $|G(w)|$ and $LZ(w)$ is known.

**Theorem 2** **(Rytter, 2003)** *For any string* $w$ *and its deterministic CFG* $G$*, the inequality* $|LZ(w)| \leq |G|$ *holds.*

Rytter introduced the *AVL-tree*, a kind of SLP such that any two leaves in the derivation tree is balanced, and showed that an AVL tree can be constructed from $LZ(w)$ by a fraction of at most $O(\lg N)$. Other two algorithms are more complicated to explain the outline. Unfortunately the algorithm proposed by Sakamoto (2005) contained an error. In 2013, this error was fixed (Jeż, 2013) and the algorithm was further improved (Jeż, 2014). Consequently we have obtained the $O(\lg N)$-approximation.

**Theorem 3 (Rytter, 2003; Charikar et al., 2005; Jeż, 2013)** *The minimum grammar compression is $O(\lg N)$ approximable.*

As fat as we know, this is the smallest approximation ratio at present. Besides, the possibility of improvement looks hopeless because the grammar compression includes the *addition chain* as a special case, whose $O(\lg N/\lg\lg N)$ ratio seems to be impossible.

As we have seen in Introduction, the direct output $G$ from the grammar compression algorithm might be larger than the raw input. Next, we focus on the optimal encoding of $G$ using the succinct data structures.

## 3. Data Structures for Optimal Encoding

### 3.1. Succinct data structures

Because we consider only the case $X \to \alpha \in P$, any grammar compression $G$ with $n$ variables can be transformed to such a restricted grammar within $2n$ variables. Then, the derivation tree is then represented by an ordered binary tree such that internal nodes are labeled by variables in $V$ and the sequence of the leaves is equal to $S$.

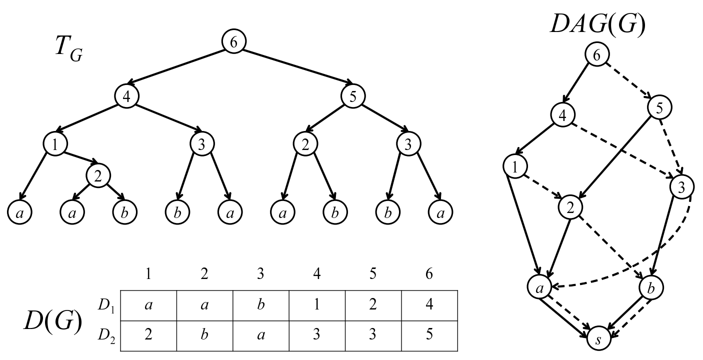
The dictionary $D$ for $P$ is a data structure if we can directly access $X_iX_j$ for any $X_k$ in case $X_k \to X_iX_j \in P$. The production rule $X_k \to X_iX_j$ can be represented by the triple $(k, i, j)$ of nonnegative integers. Thus, the set of $n$ production rules is represented by an array $D[1, 2n]$ such that $k$ indicates the production rule $(k, D[2k-1], D[2k])$.

The size of a naive representation of $D[1, 2n]$ requires $2n \lg n$ bits. The aim of encoding of CFG is to reduce its size to an asymptotically optimal one preserving the random access operation on the array. To realize such data structures, we use the *fully indexable dictionary* supporting the following three operations over $S \in \Sigma^*$ using auxiliary data structure of $o(S)$-bit[2].

- $\mathrm{access}(S, k)$ returns $S[k]$,
- $\mathrm{rank}_\sigma(S, k)$ returns the number of $\sigma$s in $S[1, k]$, and
- $\mathrm{select}_\sigma(S, k)$ returns the position in $S$ of the $k$-th $\sigma$, where $\sigma \in \Sigma$.

If $S = 10110100111$, $\mathrm{rank}_1(S, 7) = 4$, i.e., the number of 1 in $S[1, 7]$ is 4, and $\mathrm{select}_1(S, 5) = 9$, i.e., the position in $S$ of the fifth 1 is 9. When $S$ is a binary string, the response time of each rank/select is $O(1)$ (Clark, 1996; Jacobson, 1989; Munro, 1996), and for any $|\Sigma| = \sigma$, the response time is $O(\lg \sigma)$ (Grossi et al., 2003). If $S$ is a binary string, the data structure is called *succinct bit-vector* and *wavelet tree* for general $\Sigma$. We note that a weakly monotonic sequence from $[1, n]$ can be encoded in at most $2n + o(n)$ bits of space to directly access any $k$-th integer. For example, the sequence $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 2, 3, 5)$ is encoded to $S_x = 1001010110$ and for any $1 \leq k \leq 5$, we can access $x_k = \mathrm{rank}_1(S_x, \mathrm{select}_0(S_x, k))$ in $O(1)$ time.

2. $f(n) = o(g(n))$ iff $\displaystyle\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$, i.e., $g(n)$ is really smaller than $f(n)$.



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $D_1$ | $a$ | $a$ | $b$ | 1 | 2 | 4 |
| $D_2$ | 2 | $b$ | $a$ | 3 | 3 | 5 |

Figure 1: Grammar compression $G$ and its parsing tree $T_G$, DAG representation $DAG(G)$, and array representation $D(G)$, where $\Sigma = \{\mathrm{a}, \mathrm{b}\}$ and $V = \{1, 2, 3, 4, 5, 6\}$. In $DAG(G)$, the left edges are shown by solid lines. $D(G)$ itself is an implementation of the phrase dictionary.

### 3.2. Optimal encoding

Based on the bit-vector, we introduce two types of grammar encoding. One is called the *DAG representation* and the other is called the *partial parse tree*. The optimality is guaranteed by a theorem of the lower bound of the minimum bits to represent the phrase dictionary.

#### 3.2.1. DAG representation:

We represent a CFG $G$ as a DAG where $Z \to XY \in P$ is considered as two directed left edge $(Z, X)$ and right edge $(Z, Y)$, i.e., $G$ can be seen as a DAG with a single source and $|\Sigma|$ sinks. Introducing the super-sink $s$ and drawing left and right edges from any sink to $s$, we can obtain the DAG with a single source/sink equivalent to $G$. We denote the DAG as $DAG(G)$.

Renaming the variables of $DAG(G)$ in breadth-first order, we obtain an equivalent DAG whose array representation is of $D[D_1[1, n], D_2[1, n]]$ such that $X_k \to X_i X_j \in D(G)$ iff $D_1[k] = i, D_2[k] = j$ and $D_1[1, n]$ is monotonic, i.e., $D_1[i] \leq D_1[i + 1]$ (Figure 1). The monotonic sequence $D_1$ is encoded by the bit-vector $B(D_1)$ such that $B(D_1) = 0^{D_1[1]}10^{D_1[2]-D_1[1]}1 \cdots 0^{D_1[n]-D_1[n-1]}1$. We can get $D_1[k] = \mathrm{select}_1(B(D_1), k) - k$ in $O(1)$ time with $2n + o(n)$ bits of space. The remaining sequence $D_2$ is represented by GMR (Golynski et al., 2006) in $n \lg n + o(n \lg n)$ bits of space supporting $O(\lg \lg n)$ time access to any $D_2[k] = \mathrm{access}(A(D_2), k)$.

#### 3.2.2. Partial parse tree:

This concept was introduced by Rytter (2003). A partial parse tree $PTree(G)$ is obtained by the following operation: Let $T$ be the parsing tree for $w$ by $G$. If $T$ contains a maximal subtree rooted by $A \in G(V)$ appearing in $T$ at least twice, replace all occurrences of the

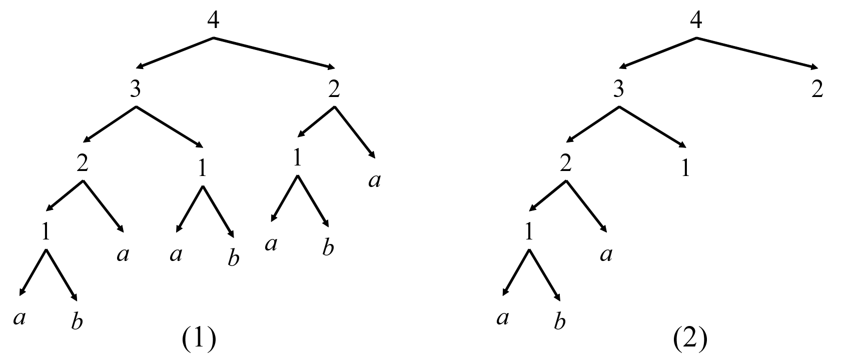


Figure 2: An example of partial parse tree and its encode by parentheses sequence for the CFG $G$ with the set of production rules, $P = \{X_4 \to X_3X_2, X_3 \to X_2X_1, X_2 \to X_1a, X_1 \to ab\}$. (**1**) the derivation tree of $G$in post-order labeling; (**2**) the partial parse tree $PTree(G)$; Then, the pair $(F, L)$ such that $F =))()()()($ and $L = aba12$ is generated. Consequently, $G$ is encoded to the sequence $(a(b)(a)(1)(2)$.

subtree by a single node labeled by $A$ except the leftmost occurrence of the subtree. Iterating this process, the rustling tree is denoted by $PTree(G)$. Figure 2 shows an example of the partial parse tree. $PTree(G)$ has $n$ internal nodes and $n + 1$ leaves, where $n = |V|$. The skeleton of $PTree(G)$ is represented by a sequence of parentheses.

Let $x_1, x_2, \ldots, x_{2n+1}$ be a sequence of nodes sorted by post-order. We represent the sequence of nodes by $2k + 1$ parentheses as follows:

$$F[i] = \begin{cases} '(' & \text{if } x_i \text{ is a leaf} \\ ')' & \text{otherwise} \end{cases} \tag{1}$$

Let $L$ be the sequence of leaf labels of $PTree(G)$ in post-order. $G$ is encoded to $(F, L)$. We estimate the bits of space required for $(F, L)$. The size of $F$ is $2n + 1$ bits. Because $L$ is the sequence over $\{1, 2, \ldots, n + |\Sigma|\}$ whose length is $n + 1$, the size of $L$ is at most $(n+1)\lceil \lg(n+|\Sigma|)\rceil$ bits. Thus, the total space for $(F, L)$ is approximately $n\lceil \lg(n+|\Sigma|)\rceil+2$ bits. A naive encoding represented by a sequence of right-hand sides of $n$ production rules requires $2n\lceil \lg(n + |\Sigma|)\rceil$ bits. Thus the succinct representation reduces the space to almost half.

We note two array $F$ and $L$ can be combined into one array such that each symbol $L[i]$ is embedded after $i$th open parenthesis of $F$. The representation of the combined array has an advantage that the decoding processing can be done in one pass over the compressed text. We can also apply simple variable-length coding like LZW (Welch, 1984) for each element of $L$ because the number of allocatable nonterminals for any leaf node is limited by the number of internal nodes that appear before the leaf node in post-order. The efficiency of compression is further improved using such variable-length coding.

#### 3.2.3. Information-theoretic lower bound:

Any SLP with $n$ variables is represented in $n \lg n + 2n + o(n)$ bits. Tabei et al. (2013b) showed that this bound is information-theoretic minimum. For a class $C$ of objects, $\lceil \lg |C| \rceil$ is the minimum bits to represent any $c \in C$, which is called the information-theoretic lower bound for $C$. To show the optimality of $n \lg n + 2n + o(n)$ bits for SLP, we need to count the number of all different SLPs with $n$ variables. To do so, the spanning tree decomposition was introduced:

*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 is also an in-branching spanning tree of $G$.*

The in-branching spanning tree consisting of the left edges (respectively the right edges) 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 1, 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 by Asai et al. (2002) and Zaki (2002) independently, 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 $T$. Thus, the problem of counting possible $G$ with $n$ variables is reduced to the problem of counting possible $T_R$ for a fixed $T_L \in \mathcal{T}_n$. Consequently the number of SLP with $n$ variables is asymptotically equal to $2^{2n}(n!)$.

**Theorem 4 (Tabei et al., 2013b)** *The information-theoretic lower bound on the minimum number of bits needed to represent an SLP with $n$ symbols is $\lg n! + 2n + o(n) \simeq n \lg n + 2n + o(n)$.*

### 3.3. Hash tables

To construct CFG $G$, we must query the naming function: When a digram $XY$ is decided to replace, it should be associated to the variable $Z$ if $Z \to XY$ is *already* created. The space for the hash table is also crucial problem. To reduce the space consumption, we introduce a method to simulate the naming function $H$ defined as follows.

For a phrase dictionary $D$ with $n$ symbols,

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

For $n = |V|$, we set a total order on $(\Sigma \cup V)^2 = \{XY \mid X, Y \in \Sigma \cup V\}$, which is represented by the range $[1, n^2]$. Then, we recursively define the wavelet tree (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 $R[\lfloor (1 + n^2) \rfloor / 2 + 1, n^2]$.

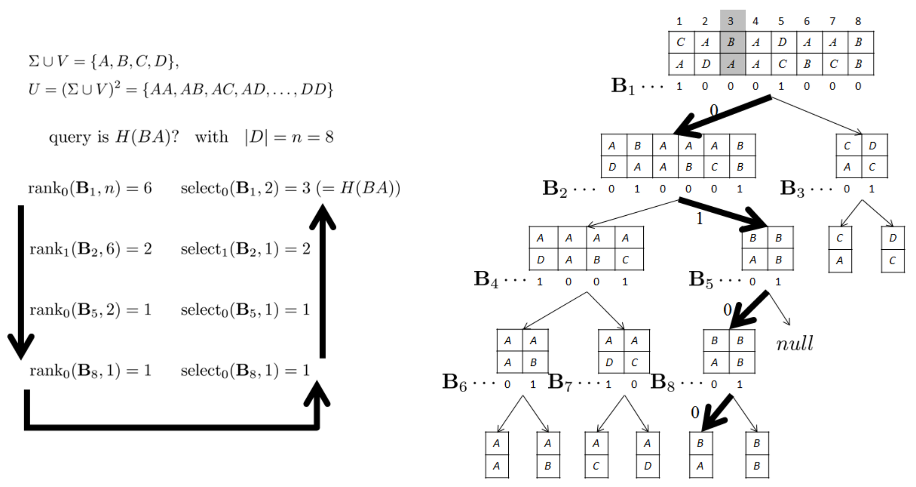

$\Sigma \cup V = \{A, B, C, D\}$,
$U = (\Sigma \cup V)^2 = \{AA, AB, AC, AD, \ldots, DD\}$
query is $H(BA)$? with $|D| = n = 8$
$\mathrm{rank}_0(\mathbf{B}_1, n) = 6$
$\mathrm{select}_0(\mathbf{B}_1, 2) = 3\ (= H(BA))$
$\mathrm{rank}_1(\mathbf{B}_2, 6) = 2$
$\mathrm{select}_1(\mathbf{B}_2, 1) = 2$
$\mathrm{rank}_0(\mathbf{B}_5, 2) = 1$
$\mathrm{select}_0(\mathbf{B}_5, 1) = 1$
$\mathrm{rank}_0(\mathbf{B}_8, 1) = 1$
$\mathrm{select}_0(\mathbf{B}_8, 1) = 1$
$\mathbf{B}_1 \cdots$ 1 0 0 0 1 0 0 0
$\mathbf{B}_2 \cdots$ 0 1 0 0 0 1
$\mathbf{B}_3 \cdots$ 0 1
$\mathbf{B}_4 \cdots$ 1 0 0 1
$\mathbf{B}_5 \cdots$ 0 1
$\mathbf{B}_6 \cdots$ 0 1
$\mathbf{B}_7 \cdots$ 1 0
$\mathbf{B}_8 \cdots$ 0 1
null

Figure 3: **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 $\mathrm{rank}_0$ and to the right child by $\mathrm{rank}_1$ on $\mathbf{B}_i$. The upward traversal is simulated by $\mathrm{select}_0$ and $\mathrm{select}_1$ as shown.

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 3, we show an example of such a data structure for a phrase dictionary $D$.

**Theorem 5 (Tabei et al., 2013b)** *The naming function for phrase dictionary $D$ over $n = |\Sigma \cup V|$ symbols can be computed by the proposed data structure $D_T$ in $O(\lg n)$ 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 $2n \lg n(1 + o(1))$ bits.*

## 4. Applications

### 4.1. Edit-sensitive parsing

First, we review the method for constructing balanced derivation tree by Cormode and Muthukrishnan (2007), called ESP (Edit-Sensitive Parsing) since all applications reviewed here are based on it, that is, we assume any SLP is parsed by ESP. When the derivation tree of $S$ and $P$ are given, it is guaranteed that $S[i, j] = P$ iff there is a sequence of subtrees in $T_P$ of length $O(\lg^* |P| \lg |P|)$ embedded adjacently to the range $S[i, j]$ of $T_S$. Using this characteristics, we can develop a self-index on grammar compression.

The basic idea is to (i) start from a string $S \in \Sigma^*$, (ii) replace as many as possible of the same digrams in common substrings by the same variables, and (iii) iterate this process until $|S| = 1$.

In each iteration, $S$ is divided into the maximal non-overlapping substrings such that $S = S_1 S_2 \cdots S_\ell$ and each $S_i$ belongs to one of three types: (1) a repetition of a symbol; (2) a substring not including a type1 substring and of length at least $\lg^* |S|$; (3) a substring being neither type1 nor type2 substrings. Substrings of $S_i$ is parsed by $A \to XY$ (2-tree) or $A \to XYZ$ (2-2-tree), where $A \to XYZ$ is further transformed to $A \to XB$ and $B \to YZ$ to obtain a binary tree.

$S_i$ is parsed according to its type. In case $S_i$ is a type1 or type3 substring, it is parsed by the typical left aligned parsing where 2-trees are built from left to right in $S_i$ and a 2-2-tree is built for the last three symbols if $|S_i|$ is odd, as follows:

- If $|S_i|$ is even, GC-ESP builds $A \to S_i[2j-1, 2j]$, $j = 1, ..., |S_i|/2$,
- Otherwise, it builds $A \to S_i[2j-1, 2j]$ for $j = 1, ..., (\lfloor |S_i|/2 \rfloor - 1)$, and builds $A \to BS_i[2j+1]$ and $B \to S_i[2j-1, 2j]$ for $j = \lfloor |S_i|/2 \rfloor$.

In case $S_i$ is a type2 substring, $S_i$ is partitioned into several substrings such that $S_i = s_1 s_2 ... s_\ell$ $(2 \le |s_j| \le 3)$ using *alphabet reduction* (Cormode and Muthukrishnan, 2007), which is detailed below. ESP builds $A \to s_j$ if $|s_j| = 2$ or builds $A \to s_j[2,3]$, $B \to s_j[1]A$ otherwise for $j = 1, ..., \ell$. After transforming $S_i$ to $S'_i$, the concatenated string $S'_i$ $(i = 1, \ldots, \ell)$ is parsed at the next level.

#### 4.1.1. Alphabet reduction:

Given a type2 substring $S$, consider $S[i]$ and $S[i-1]$ as the binary integers. Let $p$ be the position of the least significant bit in which $S[i]$ differs from $S[i-1]$, and let $bit(p, S[i]) \in \{0, 1\}$ be the value of $S[i]$ at the $p$-th position, where $p$ starts at 0. Then, $L[i] = 2p + bit(p, S[i])$ is defined for any $i \ge 2$. Since $S$ is type2, so is the resulted string $L = L[2]L[3] \ldots L[|S|]$. We note that if the number of different symbols in $S$ is $n$ which is denoted by $[S] = n$, clearly $[L] \le 2 \lg n$. Setting $S := L$, the next label string $L$ is iteratively computed until $[L] \le \lg^* |S|$, where $\lg^* n = \min\{i | \lg^{(i)} n \ge 1\}$. We can consider $\lg^* n$ as constant in practical sense, since $\lg^* n \le 5$ for $n \le 2^{65536}$. At the final $L^*$, $S[i]$ of the original $S$ is called *landmark* if $L^*[i] > \max\{L^*[i-1], L^*[i+1]\}$. After deciding all landmarks, if $S[i]$ is a landmark, it is parsed by a 2-tree or 2-2-tree.

### 4.2. Self-index

$T_S$ and $T_P$ are given. $T_P$ is divided into a sequence of maximal *adjacent* subtrees rooted by nodes $v_1, \ldots, v_k$ such that $yield(v_1 \cdots v_k) = P$, where $yield(v)$ denotes the string represented by the leaves of $v$ and $yield(v_1 \cdots v_k)$ is analogous. If $z$ is the lowest common ancestor of $v_1$ and $v_k$, which is denoted by $z = lca(v_1, v_k)$, the sequence $v_1, \ldots, v_k$ is said to be embedded into $z$, denoted by $(v_1 \cdots v_k) \prec z$. When $yield(v_1 \cdots v_k) = P$, $z$ is called an *occurrence node* of $P$.

**Definition 6 (Maruyama et al., 2013a)** *Let $L(v)$ be the variable of node $v$ and let $L(v_1 \cdots v_k)$ be the concatenation. An* evidence *of $P$ is defined as a string $Q \in (\Sigma \cup V)^*$ of length $k$*

*satisfying the following condition: There is an occurrence node $z$ of $P$ iff there is a sequence $v_1 \cdots v_k$ such that $(v_1 \cdots v_k) \prec z$, $yield(v_1 \cdots v_k) = P$, and $L(v_1 \cdots v_k) = Q$.*

This is well defined because a trivial $Q$ with $Q = P$ always exists. An evidence $Q$ transforms the problem of finding an occurrence of $P$ into that of embedding a shorter string $Q$ into $T_S$. We present an algorithm for extracting evidences.

### 4.2.1. Evidence extraction:

The evidence $Q$ of $P$ is iteratively computed from the parsing of $P$ as follows. Let $P = \alpha\beta$ for a maximal prefix $\alpha$ belonging to type1, 2 or 3. For $i$-th iteration of ESP, $\alpha$ and $\beta$ of $P$ are transformed into $\alpha'$ and $\beta'$, respectively. In case $\alpha$ is not type2, define $Q_i = \alpha$ and update $P := \beta'$. In this case, $Q_i$ is an evidence of $\alpha$ and $\beta'$ is an evidence of $\beta$. In case $\alpha$ is type2, define $Q_i = \alpha[1, j]$ with $j = \min\{p \mid p \geq \lg^* |S|,\ P[p] \text{ is landmark}\}$ and update $P := x\beta'$ where $x$ is the suffix of $\alpha'$ deriving only $\alpha[j+1, |\alpha|]$. In this case, $Q_i$ is an evidence of $\alpha[1, j]$ and $x\beta'$ is an evidence of $\alpha[j + 1, |\alpha|]\beta$. Repeating this process until $|P| = 1$, we obtain the evidence of $P$ as the concatenation of all $Q_i$. We obtain the upper bound of length $Q$.

**Theorem 7 (Maruyama et al., 2013a)** *There is an evidence $Q$ of $P$ such that $Q = Q_1 \cdots Q_k$ where $Q_i \in q_i^+$ ($q_i \in \Sigma \cup V$, $q_i \neq q_{i+1}$) and $k = O(\lg |P| \lg^* |S|)$.*

### 4.2.2. Counting, locating, and extracting:

A node $z$ in $T_S$ is an occurrence node of $P$ iff there is a sequence $v_1, \ldots, v_k$ such that $(v_1, \ldots, v_k) \prec z$ and $L(v_1 \cdots v_k) = Q$. It is sufficient to adjacently embed all subtrees of $v_1, \ldots, v_k$ into $T_S$. We recall the fact that the subtree of $v_1$ is left adjacent to that of $v_2$ iff $v_2$ is a leftmost descendant of $right\_child(lra(v_1))$ where $lra(v)$ denotes the *lowest right ancestor of* $v$, i.e., the lowest ancestor of $v$ such that the path from $v$ to it contains at least one left edge. Because $z = lra(v_1)$ is unique and the height of $T_S$ is $O(\lg |S|)$, we can check whether $(v_1, v_2) \prec z$ in $O(\lg |S|)$ time. Moreover, $(v_1, v_2, v_3) \prec z'$ iff $(z, v_3) \prec z'$ (possibly $z = z'$). Therefore, when $|Q_i| = 1$ for each $i$, we can execute the embedding of whole $Q$ in $t = O(\lg |P| \lg |S| \lg^* |S|)$ time. For general case of $Q_i \in q_i^+$, the same time complexity $t$ is obtained.

**Theorem 8 (Maruyama et al., 2013a; Takabatake et al., 2014a)** *Let $|S| = N$, $|P| = m$, and $|G| = n$. Counting time of ESP-index is $O((m + occ \lg m \lg N) \lg\lg n \lg^* N)$ with $2n + n \lg n + o(n \lg n)$ bits of space. With auxiliary $n \lg N + o(n)$ bits of space, ESP-index supports locating in the same time complexity and also supports extracting in $O((m + \lg N) \lg\lg n)$ time where occ is the frequency of the largest embedded subtree.*

The earlier version of ESP-index (Maruyama et al., 2013a) was implemented by LOUDS (Delpratt et al., 2006) and permutation (Munro et al., 2003) with the time-space trade-off. The space was improved by Takabatake et al. (2014a) using GMR (Golynski et al., 2006).

| compression time | working space (bits) | Ref. |
|---|---|---|
| $O(N/\alpha)$ expected | $(3+\alpha)n\lg(n+\lvert\Sigma\rvert)$ | Maruyama et al. (2012) |
| $O(N/\alpha)$ expected | $(\frac{11}{4}+\alpha)n\lg(n+\lvert\Sigma\rvert)$ | Takabatake et al. (2012) |
| $O(N\lg n)$ | $2n\lg n(1+o(1))+2n\lg\rho\ (\rho\leq 2\sqrt{n})$ | Tabei et al. (2013a) |
| $O(\frac{N\lg n}{\alpha\lg\lg n})$ expected | $(1+\alpha)n\lg(n+\lvert\Sigma\rvert)+n(3+\lg(\alpha n))$ | Maruyama et al. (2013b) |
| $O(N\lg n)$ | $2n\lg n(1+o(1))+2n$ | Maruyama et al. (2013b) |

Table 1: Comparison with online algorithms. $N$: the length of string, $n$: the number of generated rules, and $\alpha\in(0,1]$ (i.e., the load factor of hash tables). The size of the auxiliary index for efficient substring decoding is excluded. The expected time complexities are due to the use of a hash function.

## 4.3. Online algorithm

There is a strong demand to manage large-scale and highly repetitive text collections in an online fashion. Grammar compression reveals high compressive and processing abilities for highly repetitive texts in pattern matching (Tiskin, 2011; Yamamoto et al., 2011), edit-distance computation (Hermelin et al., 2009), $q$-gram mining (Goto et al., 2013) and mining characteristic substrings (Inenaga and Bannai, 2009; Matsubara et al., 2009), etc. Basically, existing methods first build a complete CFG from an input text, and then encode it into a compact representation. A crucial drawback of those methods is to require a large working space consumed for building a CFG and its encoding. Even worse, they can not deal with stream data because of their static property.

Maruyama et al. (2012) solved the inefficiency problem of large working space and the static property by introducing an online grammar compression called *online LCA* (OLCA). Although OLCA achieves a good worst-case approximation ratio of $O(\lg^2 N)$ to the smallest CFG for an input string of length $N$, it has a serious issue of large working space and its inability of direct encoding of an SLP into a succinct representation. Later, Takabatake et al. (2012) presented an online encoding scheme of an SLP of $n$ variables built from OLCA into a succinct representation achieving $\frac{7}{4}n\lg n+4n+o(n)$ bits, which was still larger than the information-theoretic lower bound (Tabei et al., 2013a). Moreover, they did not present a space-efficient *reverse dictionary*, a crucial data structure for checking whether or not a production rule in an SLP already exists in execution, which has been implemented using a chaining hash table having a load factor $\alpha$. Though this scheme is fully-online, its working space is larger than an information-theoretic lower bound. Since available data of highly repetitive texts is ever increasing, developing a fully-online grammar compression for building an SLP using the minimum space remains a challenge.

Maruyama et al. (2013b) presented the fully-online grammar compression building the SLP and directly encoding it into a succinct representation in an online manner. This algorithm called *fully-online LCA* (FOLCA), which is a modification of OLCA that builds the post order SLP (POSLP). A major advantage of a POSLP is enabling a direct encoding into a succinct representation, while keeping the approximation ratio $O(\lg^2 N)$ of OLCA. Table 1 summarizes these results.

| | Appro. ratio | Time | Space | Algorithm |
|---|---|---|---|---|
| SNN | $O(\lg N \lg^* N)$ | $O(N^{O(1)} + N\text{polylog}(N))$ | $O(N^{O(1)})$ | Offline |
| Shapira & Storer | $O(\lg N)$ | $O(N^2)$ | $O(N \lg N)$ | Offline |
| ESP | $O(\lg N \lg^* N)$ | $O(N \lg^* N/\alpha)$ | $N \lg \lvert\Sigma\rvert$ $+n(\alpha+3)\lg(n+\lvert\Sigma\rvert)$ | Offline |
| OESP | $O(\lg^2 N)$ | $O(\frac{N \lg N \lg n}{\alpha \lg\lg n})$ | $n(\alpha+1)\lg(n+\lvert\Sigma\rvert)$ $+n(5+\lg(\alpha n))+o(n)$ | Online |

Table 2: Comparison with pattern matching methods for EDM. The table summaries the approximation ratio to EDM, computation time and space. The space for ESP and OESP is presented in bits. SNN (Muthukrishnan and Sahinalp, 2000); ESP (Cormode and Muthukrishnan, 2007); Shapira and Storer (Shapira and Storer, 2007).

### 4.4. Similarity metric

Streaming text data appears in many application domains of information retrieval. For example recent sequencing technologies enable us to sequence individual genomes in a short time, which resulted in generating a large collection of genome data. There is therefore a strong incentive to develop a powerful method for similarity metric on a large cellection of data.

Originally, ESP (Cormode and Muthukrishnan, 2007) was invented to approximately compute the *Edit distance with moves (EDM)*, which is a string-to-string distance measure that includes substring moves in addition to insertions and deletions to turn one string to the other in a series of editing operations. The distance measure is motivated in error detections, e.g., insertions and deletions on lossy communication channels (Levenshtein, 1996), typing errors in documents (Crochemore and Rytter, 1994) and evolutionary changes in biological sequences (Durbin et al., 1998). Computing an optimum solution of EDM is intractable due to its NP-completeness (Muthukrishnan and Sahinalp, 2000). Therefore, researchers have paid considerable efforts to develop efficient approximation algorithms that are only applicable to an offline case where a whole text is given in advance (Table 2). Early results include the reversal model (Kececioglu and Sankoff, 1993; Bafna and Pevzner, 1996) which takes a substring of unrestricted size and replaces it by its reverse in one operation. Muthukrishnan and Sahinalp (2000) proposed an approximate nearest neighbor considered as a sequence comparison with block operations. Recently, Shapira and Storer (2007) proposed a polylog time algorithm with $O(\lg N)$ approximation ratio for the length $N$ of an input text.

ESP (Cormode and Muthukrishnan, 2007) is an efficient parsing algorithm developed for approximately computing EDM between strings in an offline setting. ESP builds from a given string a parse tree that guarantees upper bounds of parsing discrepancies between different appearances of the same substring, and then it represents the parse tree as a vector each dimension of which represents the frequency of the corresponding node label in a parse tree. $L_1$ distance between such characteristic vectors for two strings can approximate the EDM. Although ESP has an efficient approximation ratio $O(\lg N \lg^* N)$ and runs fast in $O(N \lg^* N/\alpha)$ time for a parameter $\alpha \in (0,1]$ for hash tables, its applicability is limited to an offline case. For applications in web mining and Bioinformatics, computing an EDM of massive streaming text data has ever been an important task. A challenge is to develop a scalable online pattern matching for EDM.

An online pattern matching for EDM (Takabatake et al., 2014b) was recently presented by introducing a novel succinct representation of *post-order unary degree sequence (POUDS)*. The method is an online version of ESP named *online ESP (OESP)* that (i) builds a parse tree for a streaming text in an online manner, (ii) computes characteristic vectors for a substring at each position of the streaming text and a query, and (iii) computes the $L_1$ distance between each pair of characteristic vectors. The working space does not depend on the length of text but the size of the grammar compression. To keep the working space smaller, OESP builds a parse tree from a streaming text and directly encodes it into a succinct representation leveraging the idea in the grammar compression (Maruyama et al., 2013b; Maruyama and Tabei, 2014) and a dynamic succinct tree (Navarro and Sadakane, 2014).

## 5. Open Problem

We have reviewed the grammar compression and its applications. Finally, we propose two open problems concerning stream grammar compression.

One is the online construction of self-index, i.e., given a self-index for $S$, we must dynamically update it for $S\alpha$. If the index is based on BWT (Burrows and Wheeler, 1994), this problem depends on the hardness of updating a sorted array for a new entry. Theoretically, it is solvable using the compressed random access memory (Jansson et al., 2012), the implementation is, however, very hard. In case of grammar compression, there is a possibility of the online self-index thanks to the stability of the derivation tree. When $T_S$ is constructed, we do not edit the $T_S$ to complete the derivation tree $T_{S\alpha}$. Currently, there is no algorithm for this problem.

The other interesting challenge is the speeding-up of data transmission by grammar compression. In the communication complexity, it is a traditional problem to transmit data by compression. However, taking account of the overhead of compression and decompression, this mechanism does not work well. For this problem, a simple CFG construction (Yamagiwa and Sakamoto, 2013) was proposed and its prototype was demonstrated. As shown in this study, the hardware implementation is a hopeful application of grammar compression.

If you challenge such problems, you should deeply learn about the hardness and approximability of grammar compression. Lehman's PhD thesis (Lehman, 2002) is the best textbook. Further, you can obtain the basic knowledge of succinct data structures by Navarro's invited paper (Navarro, 2012).

## Acknowledgment

I thank Yoshimasa Takabatake, Shirou Maruyama, Yasuo Tabei, Kunihiko Sadakane, Tetsuji Kuboyama, Shinich Yamagiwa, Rajiv Raman, and Gonzalo Navarro for their discussion.

## References

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.

V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25:272–289, 1996.

D.A. Benoit, E.D. Demaine, J.I. Munro, R. Raman, V. Raman, and S.S. Rao. Representing trees of higher degree. *Algorithmica*, 43:275–292, 2005.

M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical report, TRep. 124, DEC, 1994.

M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inform. Theory*, 51(7):2554–2576, 2005.

R. Cilibrasi and P.M.B. Vitanyi. Clustering by compression. *IEEE Trans. Inform. Theory*, 51(4):1523–1545, 2005.

D. Clark. *Compact Pat Trees.* PhD thesis, University of Waterloo, 1996.

F. Claude and G. Navarro. Self-indexed grammar-based compression. *Fundam. Inform.*, 111(3):313–337, 2011.

G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algor.*, 3(1):Article 2, 2007.

M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.

O. Delpratt, N. Rahman, and R. Raman. Engineering the louds succinct tree representation. In *WEA*, pages 134–145, 2006.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Algor.*, 3(2), 2007.

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.

K. Goto, H. Bannai, S. Inenaga, and M. Takeda. Fast q-gram mining on slp compressed strings. *J. Discrete Algorithms*, 18:89–99, 2013.

R. Grossi, A. Gupta, and J.S. Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 636–645, 2003.

D. Hermelin, G.M. Landau, S. Landau, and O. Weimann. A unified algorithm for accelerating edit-distance computation via text-compression. In *STACS*, pages 26–28, 2009.

S. Inenaga and H. Bannai. Finding characteristic substrings from compressed texts. In *PSC*, pages 40–54, 2009.

G. Jacobson. Space-efficient static trees and graphs. In *FOCS*, pages 549–554, 1989.

J. Jansson, K. Sadakane, and W.K. Sung. Cram: Compressed random access memory. In *ICALP*, volume 1, pages 510–521, 2012.

A. Jeż. Approximation of grammar-based compression via recompression. In *CPM2013*, pages 165–176, 2013.

A. Jeż. A really simple approximation of smallest grammar. In *CPM2014*, pages 182–191, 2014.

M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nordic J. Comp.*, 4(2):172–186, 1997.

J. D. Kececioglu and D. Sankoff. Exact and approximation algorithms for the inversion distance between two chromosomes. In *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, pages 87–105, 1993.

S. Kreft and G. Navarro. Self-indexing based on LZ77. In *CPM*, pages 41–54, 2011.

N.J. Larsson and A. Moffat. Offline dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.

E. Lehman. *Approximation Algorithms for Grammar-Based Compression.* PhD thesis, MIT, 2002.

E. Lehman and A. Shelat. Approximation algorithms for grammar-based compression. In *SODA*, pages 205–212, 2002.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1996.

M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitanyi. The similarity metric. *IEEE Trans. Inform. Theory*, 50(12):3250–3264, 2004.

H.-I. Lu and C.-C. Yeh. Balanced parentheses strike back. *ACM Transactions on Algorithms*, 4(3):Article No 28, 2008.

S. Maruyama and Y. Tabei. Fully-online grammar compression in constant space. In *Proceedings of Data Compression Conference*, pages 218–229, 2014.

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. To appear in *Journal of Discrete Algorithms.*

S. Maruyama, H. Sakamoto, and M. Takeda. An online algorithm for lightweight grammar-based compression. *Algorithms*, 5(2):213–235, 2012.

S. Maruyama, M. Nakahara, N. Kishiue, and H. Sakamoto. ESP-Index: A Compressed Index Based on Edit-Sensitive Parsing. *Journal of Discrete Algorithms*, 18:100–112, 2013a.

S. Maruyama, Y. Tabei, H. Sakamoto, and K. Sadakane. Fully-online grammar compression. In *SPIRE2013*, pages 218–229, 2013b.

W. Matsubara, S. Inenaga, A. Ishino, A Shinohara, T. Nakamura, and K. Hashimoto. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. *Theoretical Computer Science*, 410(8-10):900–913, 2009.

J.I. Munro. Tables. In *FSTTCS*, pages 37–42, 1996.

J.I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001.

J.I. Munro, R. Raman, V. Raman, and S.S. Rao. Succinct representations of permutations. In *ICALP*, pages 345–356, 2003.

S Muthukrishnan and S. C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. In *Proceedings of the 32nd annual ACM symposium on Theory of computing*, pages 416–424, 2000.

M. Nakahara, S. Maruyama, T. Kuboyama, and H. Sakamoto. Scalable detection of frequent substrings by grammar-based compression. *IEICE Transactions*, 96-D(3):457–464, 2013.

G. Navarro. Wavelet trees for all. In *CPM*, pages 2–26, 2012.

G. Navarro and K. Sadakane. Fully-functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3):16, 2014. A preliminary version appeared in SODA 2010.

R. Raman, V. Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *SODA*, pages 233–242, 2002.

W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.

K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *J. Algorithms*, 48(2):294–313, 2003.

K. Sadakane and G. Navarro. Fully-functional succinct trees. In *SODA*, pages 134–149, 2010.

H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *J. Discrete Algorithms*, 3(2-4):416–430, 2005.

H. Sakamoto, S. Maruyama, T. Kida, and S. Shimozono. A space-saving approximation algorithm for grammar-based compression. *IEICE Trans. Inform. and Systems*, E92-D (2):158–165, 2009.

D. Shapira and J. A. Storer. Edit distance with move operations. *Journal of Discrete Algorithms*, 5:380–392, 2007.

Y. Tabei, Y. Takabatake, and H. Sakamoto. A succinct grammar compression. In *CPM*, 2013a.

Y. Tabei, Y. Takabatake, and H. Sakamoto. A succinct grammar compression. In *CPM2013*, pages 235–246, 2013b.

Y. Takabatake, Y. Tabei, and H. Sakamoto. Variable-length codes for space-efficient grammar-based compression. In *SPIRE*, pages 398–410, 2012.

Y. Takabatake, Y. Tabei, and H. Sakamoto. Improved esp-index: A practical self-index for highly repetitive texts. In *SEA2014*, pages 338–350, 2014a.

Y. Takabatake, Y. Tabei, and H. Sakamoto. Online pattern matching for string edit distance with moves. In *SPIRE2014, to appear*, 2014b.

A. Tiskin. Towards approximate matching in compressed strings. In *CSR*, pages 401–414, Berlin, Heidelberg, 2011. Springer-Verlag.

T.A. Welch. A technique for high performance data compression. *IEEE Comput.*, 17:8–19, 1984.

S. Yamagiwa and H. Sakamoto. A reconfigurable stream compression hardware based on static symbol-lookup table. In *BPOE2013*, pages 86–93, 2013.

T. Yamamoto, H. Bannai, S. Inenaga, and M. Takeda. Faster subsequence and don't-care pattern matching on compressed texts. In *CPM*, volume 6661, pages 309–322, 2011.

M.J. Zaki. Efficiently mining frequent trees in a forest. In *KDD*, pages 71–80, 2002.

J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory*, 23(3):337–343, 1977.

J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inform. Theory*, 24(5):530–536, 1978.