Grammar compression with probabilistic context-free grammar

Hiroaki Naganuma*, Diptarama Hendrian*, Ryo Yoshinaka*, Ayumi Shinohara*, and Naoki Kobayashi†

*Tohoku University, †The University of Tokyo

Abstract: We propose a new approach for universal lossless text compression, based on grammar compression. In the literature, a target string T has been compressed as a context-free grammar G in Chomsky normal form satisfying $L(G) = T$. Such a grammar is often called a straight-line program (SLP). In this work, we consider a probabilistic grammar G that generates T, but not necessarily as a unique element of $L(G)$. In order to recover the original text T unambiguously, we keep both the grammar G and the derivation tree of T from the start symbol in G, in compressed form. We show some simple evidence that our proposal is indeed more efficient than SLPs for certain texts, both from theoretical and practical points of view.

Existing approaches for grammar compression

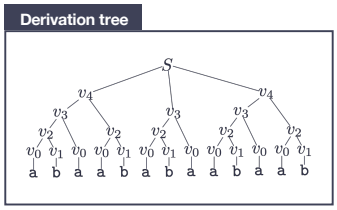
Approach 1: Universal

- Given a text T, an encoder constructs CFG G such that $L(G) = \{T\}$.

Grammar G

$v_0 \to$ a
$v_1 \to$ b
$v_2 \to v_0v_1$
$v_3 \to v_2v_0$
$v_4 \to v_3v_2$
$S \to v_4v_3v_4$

Derivation tree



$|G| = 11$

An encoder tries to construct the minimum CFG. (**The smallest grammar problem** [Charikar+, 2005])

$|G|$: A grammar size of CFG G. It is defined as the total length of strings on the right hand sides of all production rules.

Approach 2: Domain-specific

- A fixed CFG G is used when an encoder compresses an input text T such that $T \in L(G)$
- A compressed data is a derivation sequence of T in G which is a sequence of production rules in G.

Grammar G

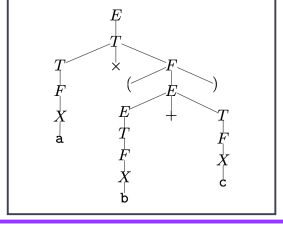
$E \to E+T \mid E-T \mid T$
$T \to T \times F \mid T \div F \mid F$
$F \to (E) \mid X$
$X \to$ a $\mid$ b $\mid$ c $\mid$ d

$E \Rightarrow T \Rightarrow T{\times}F \Rightarrow F{\times}F \Rightarrow X{\times}F \Rightarrow \cdots$

$E \to T$ $T \to T{\times}F$ $T \to F$ $F \to X$

Derivation sequence

Derivation tree



Our proposal framework of grammar compression using PCFG

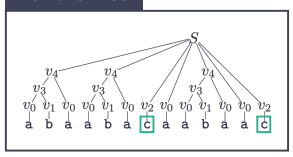
Disadvantage of approach 1: sensitive to noise

The example of CFG G' constructed from T' =abaabacaabaac:

Grammar G'

$v_0 \to$ a
$v_1 \to$ b
$v_2 \to$ c
$v_3 \to v_0v_1$
$v_4 \to v_3v_0$
$S \to v_4v_4v_2v_0v_4v_0v_2$

Derivation tree



$|G'| = 14$

Grammar becomes bigger by noises in T'

Disadvantage of approach 2: no universality

We cannot compress a text T such that $T \notin L(G)$ effectively.

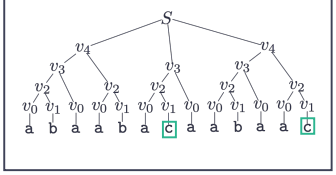
Idea for improvement

- Given a text T, an encoder constructs CFG G such that $T \in L(G)$.
- A compressed data consists of G and a derivation sequence of T in G.
- A derivation sequence is encoded by **arithmetic coding**.

Grammar

$v_0 \to$ a
$v_1 \to$ b $\mid$ c
$v_2 \to v_0v_1$
$v_3 \to v_2v_0$
$v_4 \to v_3v_2$
$S \to v_4v_3v_4$

Derivation tree



Effectiveness of our proposed scheme

Theoretical result

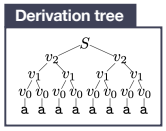
Comparison of SLP compression and our framework for $T = a^n$ where $n = 2^m$

SLP

Grammar

$v_0 \to$ a
$v_1 \to v_0v_0$
$v_2 \to v_1v_1$
$S \to v_2v_2$

Derivation tree



- $\log_2 m$ bits for the m nonterminals
- $m\log_2 m$ bits for the m rules

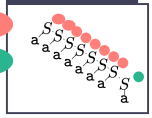
$\Omega(\log n \log\log n)$ bits

Our framework

Grammar

$S \to$ aS (Rule 1)
$S \to$ a (Rule 2)

Derivation tree



- Grammar size: constant to n
- Arithmetic code size: $-\log_2 \left(\frac{n-1}{n}\right)^{n-1} \frac{1}{n}$ bits

$\Theta(\log n)$ bits

Experimental result

Compressing "noisy" 20th Fibonacci strings (10946 bytes) by ideal grammars in our framework, grammars output by our prototype compression algorithm, and some existing compressors

Noise types

- Type 0: replace a with b or b with a
- Type k: replace a letter with one of k new letters $c_1, c_2, \cdots, c_k$

T =abaababaabaab
Type 0: aaabbababbaab
Type 2: axaybabaxbaab

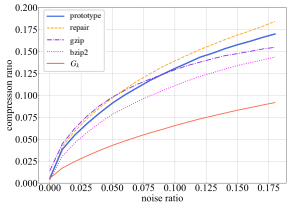
Ideal Grammar G_k based on Fibonacci grammar

For Type 0

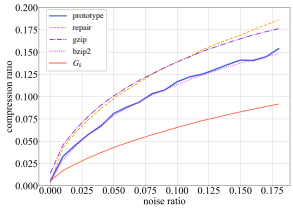
$v_0 \to$ b $\mid$ a
$v_1 \to$ a $\mid$ b
$v_2 \to v_1v_0$
$v_3 \to v_2v_1$
$\vdots$
$S \to v_{i-1}v_{i-2}$

For Type k

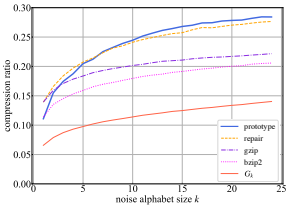
$v_0 \to$ b $\mid c_1 \mid c_2 \mid \cdots \mid c_k$
$v_1 \to$ a $\mid c_1 \mid c_2 \mid \cdots \mid c_k$
$v_2 \to v_1v_0$
$v_3 \to v_2v_1$
$\vdots$
$S \to v_{i-1}v_{i-2}$



Type 0



Type 1



Type k ($1 \le k \le 24$)