On the Complexity of Optimal Grammar-Based Compression

By Jan Arpe and Rudiger Reischuk

Schriftenreihe der Institute fur Informatik/Mathematik

Jason Lustig

Grammar-Based Compression: What??

- Encoding: Create a context-free grammar for a string of characters or bits
- Code the grammar and transfer it
- Decoding: Parse and expand the grammar

A short example

X = "DOG EAT DOG"

T1 → D
T2 → O
T3 →G
T4 → [SPACE]
T5 →E
T6 →A
T7 →T

V1 → T1T2T3
V2 → T5T6T7
S → V1T4V2T4V1

Relationship to sliding window and other methods

- Sliding window methods can be seen as a specific case and coding of grammar-based compression
- You are doing text-replacement... isn't it the same thing?
- Sort of like the arithmetic coding of grammars - you can start decompressing before the whole input shows up

Goals of grammar-based compression

- It must be deterministic -- i.e. you can only get one expansion from a grammar
- It should be as small as possible

This is the hard part

- Fairly certain that finding the *Minimum Grammar Compression* (MGC) is quite hard
- It is NP-complete, in fact, when restricted to alphabets of size >= 3
- However, they are not sure if it is NP-hard in binary encodings (We call this 2MGC)

So what do we do?

- Try to *approximate* the minimum grammar
- Smallest grammars known so far are of size at most O(log n), but not sure this is the best possible

What is new here?

- Authors try to show relationship between grammars on strings with a library of arbitrary size and those with a finite size
- By making a block coding from a string in one arbitrary alphabet to a finite one, they show how the size of a grammar for the coding size is related to that of the original string
- This reduces case of arbitrary alphabets to finite ones

It's all greek to me

- τ : Finite or infinite alphabet
- Σ : Finite alphabet
- φ : Block coding, $\varphi(x \mid x \in \tau^*) = x \dashrightarrow \Sigma^*$
- G_x : Grammars for x. {Σ, V = nonterminals, P = derivations, S = start symbols}
- $m(G_x)$: size of the grammar
- $m^*(G_x)$: size of smallest grammar

Coding grammar and string grammar

- Let $x \in \tau^*$, φ : l-block code, $\tau^*$to Σ^*
- Grammar for x has size m(x), grammar for $\varphi(\tau_x)$ has size m(φ)
- Grammar for φ(x) has size m(φ(x)) <= m (x)+m(φ)

An example

$\tau = \{0, 1, 2, 3, 4, 5, 6, 7\}; |\tau| = 8$

$\Sigma = \{0, 1\}; |\Sigma| = 2$

$\varphi : \tau_x{}^* \to \Sigma^*$:

$0 \to 000$		$4 \to 100$	
$1 \to 001$		$5 \to 101$	
$2 \to 010$		$6 \to 110$	
$3 \to 011$		$7 \to 111$	

φ is an *l-block coding* where l=3

$x \in \tau^*$, $\varphi(x) \in \Sigma^*$

x = "2 5 4 7 6 1 2 2 2 5"

φ(x) = "010 110 100 111 110 001 010 010 010 110"

G_x

Terminals	Nonterminals	Start
$T_2 \rightarrow 2$ $T_5 \rightarrow 5$ $T_4 \rightarrow 4$ $T_7 \rightarrow 7$ $T_6 \rightarrow 6$ $T_1 \rightarrow 1$	$NT_0 \rightarrow T_2T_5$ $NT_1 \rightarrow T_4T_7$ $NT_2 \rightarrow T_6T_1$ $NT_3 \rightarrow T_2T_2$ $NT_4 \rightarrow NT_0NT_1$ $NT_5 \rightarrow NT_2NT_3$ $NT_6 \rightarrow NT_4NT_5$	$S_x \rightarrow NT_6NT_4$

$m(x) = |Terminals| + |Nonterminals| + |Start| = 14$

G_φ

Terminals	Nonterminals	Start
$T_2 \rightarrow 010$ $T_5 \rightarrow 101$ $T_4 \rightarrow 100$ $T_7 \rightarrow 111$ $T_6 \rightarrow 110$ $T_1 \rightarrow 001$		$S_2 \rightarrow T_2$ $S_5 \rightarrow T_5$ $S_4 \rightarrow T_4$ $S_7 \rightarrow T_7$ $S_6 \rightarrow T_6$ $S_1 \rightarrow T_1$

$m(\varphi) = |\text{Terminals}| + |\text{Nonterminals}| + |\text{Start}| = 12$

What we can do

- Make a grammar for $\varphi(x)$, of size $m(\varphi(x))$
- $m(\varphi(x)) \leq m(x) + m(\varphi)$
- From $G_{\varphi(x)}$ authors construct another grammar for x, $m(x) \leq 2*l*m(\varphi(x))$
- If φ is overlap-free, $m(x) <= 2*m(\varphi(x))$
- They also show that this holds for $m^*(x)$

More on binary alphabets

- Authors are interested in binary because it is the most practical
- Take an l-block code $\varphi : \tau_x^* \rightarrow \{0, 1\}^*$
- $m^*(x) >= 1/24 * l * m^*(\varphi(x))$
- In other words:
  - $m^*(\varphi(x)) \leq 24/l * |\tau|$

Bounded v. Unbounded

- For all $\varepsilon > 0$, there is a natural *n* such that if $|x| >= n$, then for any G_x of size $m(x)$, we can make a grammar for $\varphi(x)$ of size $m(\varphi(x)) \leq (12 + \varepsilon)m(x)$

- Authors can then take this and make another G_x of size $m(x) \leq 2m(\varphi(x))$

- This shows us that the size of grammars for bounded v. unbounded alphabets only differs by constant factors

So what?

- This implies that if MGC cannot be done within constant factors for arbitrary strings, it can't be done for binary either
- Also, they showed that for unbounded and finite alphabets the grammars are related by constants
- Needs additional research - find optimal grammar-based compression for a set of strings

?