On the Complexity of Grammar-Based Compression over Fixed Alphabets

Katrin Casel[1], Henning Fernau[1], Serge Gaspers[2], Benjamin Gras[3], **Markus L. Schmid**[1]

[1] Trier University, Germany
[2] Data61, CSIRO, Australia
[3] École Normale Superieure de Lyon, France

Highlights 2016
of Logics, Games and Automata

The Problem

Grammar-Based Compression

Input: word w,
Output: context-free grammar for $\{w\}$.

The Problem

Grammar-Based Compression

Input: word w,
Output: context-free grammar for $\{w\}$.

Shortest grammar — Optimal compression

Short grammar — Approximated compression

Short/shortest grammar with ... — Parameterised variants
... only $\leq k$ rules,
... derivations tree with $\leq k$ levels,
... length of right sides of rules $\leq k$,

Motivation

Algorithmics on compressed strings!
(i. e., solving problems directly on compressed data)

Grammars

- cover many compression schemes from practice (e. g., Lempel-Ziv),
- are mathematically easy to handle,
- allow solving of basic problems (comparison, pattern matching, membership in a regular language, retrieving subwords) efficiently.

Motivation

Algorithmics on compressed strings!
(i. e., solving problems directly on compressed data)

Grammars

- cover many compression schemes from practice (e. g., Lempel-Ziv),
- are mathematically easy to handle,
- allow solving of basic problems (comparison, pattern matching, membership in a regular language, retrieving subwords) efficiently.

Grammar-based compression has

- applications in combinatorial group theory, comput. topology,
- been extended to more complicated objects, e. g., trees, $2D$ words.

Hardness of the Problem

Finding a minimal grammar can be surprisingly difficult
(even for a fixed and simply structured word).

We gave up on determining a smallest grammar for

$$10\,100\,1000\,10000\,100000\ldots 10^n \qquad \text{with } n = 2^k.$$

Shortest Grammar Problem

Shortest Grammar Problem SGP

Instance: A word w and a $k \in \mathbb{N}$.

Question: $\exists$ grammar G with $L(G) = \{w\}$ and $|G| \leq k$?

Shortest Grammar Problem

Shortest (1-Level) Grammar Problem (1−)SGP

Instance: A word w and a $k \in \mathbb{N}$.
Question: $\exists$ (1-level) grammar G with $L(G) = \{w\}$ and $|G| \leq k$?

1-level grammar:
every rule $A \to \alpha$ (except start rules) satisfies $\alpha \in \Sigma^*$.

Known Results

Main open problem:

Is SGP (or 1-SGP) NP-hard?

Known Results

Main open problem:

Is SGP (or 1-SGP) NP-hard?

Known:

SGP is NP-complete for unbounded alphabets
(artificial problem, simple result)

Known Results

Main open problem:

Is SGP (or 1-SGP) NP-hard?

Known:

SGP is NP-complete for unbounded alphabets
(artificial problem, simple result)

Wrongly claimed in literature:

SGP is NP-complete for alphabets of size 3

Main Result

Theorem

SGP *is* NP*-complete, even for alphabets of size* 24*.*
(for alphabets of size 5 *in the* 1*-level case)*

Main Result

Theorem

SGP *is* NP*-complete, even for alphabets of size* 24*.*
(for alphabets of size 5 *in the* 1*-level case)*

Proof: Terribly technical reduction (simply horrible!), but why?
Fixed alphabet $\Rightarrow$ codewords.
We need to know what a best grammar does to the codewords.

Reduction

Graph $G \Rightarrow$ word uvw

$$\mathbf{u} = \prod_{\mathbf{j=0}}^{\mathbf{6}} \left(\prod_{\mathbf{i=1}}^{\mathbf{14n}} (\langle \mathbf{i} \rangle_{\diamond} \ \langle \mathbf{M(i+j, 14n)} \rangle_{\mathbf{v}}) \right) \ \$_1$$

$$\mathbf{v} = \prod_{\mathbf{i=1}}^{\mathbf{n}} (\# \ \langle \mathbf{7i + C_v(i)} \rangle_{\mathbf{v}} \ \cent_1 \ \langle \mathbf{7i-1} \rangle_{\diamond}) \ \$_2 \prod_{\mathbf{i=1}}^{\mathbf{n}} (\# \ \langle \mathbf{7i + C_v(i)} \rangle_{\mathbf{v}} \ \cent_2 \ \langle \mathbf{7i-2} \rangle_{\diamond}) \ \$_3$$
$$\prod_{\mathbf{i=1}}^{\mathbf{n}} (\langle \mathbf{7i + C_v(i)} \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7i-2} \rangle_{\diamond} \ \cent_1) \ \$_4 \prod_{\mathbf{i=1}}^{\mathbf{n}} (\langle \mathbf{7i + C_v(i)} \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7i-1} \rangle_{\diamond} \ \cent_2) \ \$_5$$
$$\prod_{\mathbf{i=1}}^{\mathbf{n}} (\# \ \langle \mathbf{7i + C_v(i)} \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7i} \rangle_{\diamond}) \ \$_6$$

$$\mathbf{w} = \prod_{\mathbf{i=1}}^{\mathbf{m-1}} (\# \ \langle \mathbf{7j_{2i-1} + C_v(j_{2i-1})} \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7j_{2i} + C_v(j_{2i})} \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7i + C_e(v_{j_{2i}}, v_{j_{2i+1}})} \rangle_{\diamond})$$
$$\# \ \langle \mathbf{7j_{2m-1} + C_v(j_{2m-1})} \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7j_{2m} + C_v(j_{2m})} \rangle_{\mathbf{v}} \ \#$$

Bounded Number of Non-Terminals

Theorem

Let $w \in \Sigma^*$ and $k \in \mathbb{N}$,

- *a grammar that is minimal among all grammars with at most k rules,*
- *a 1-level grammar that is minimal among all 1-level grammars with at most k rules*

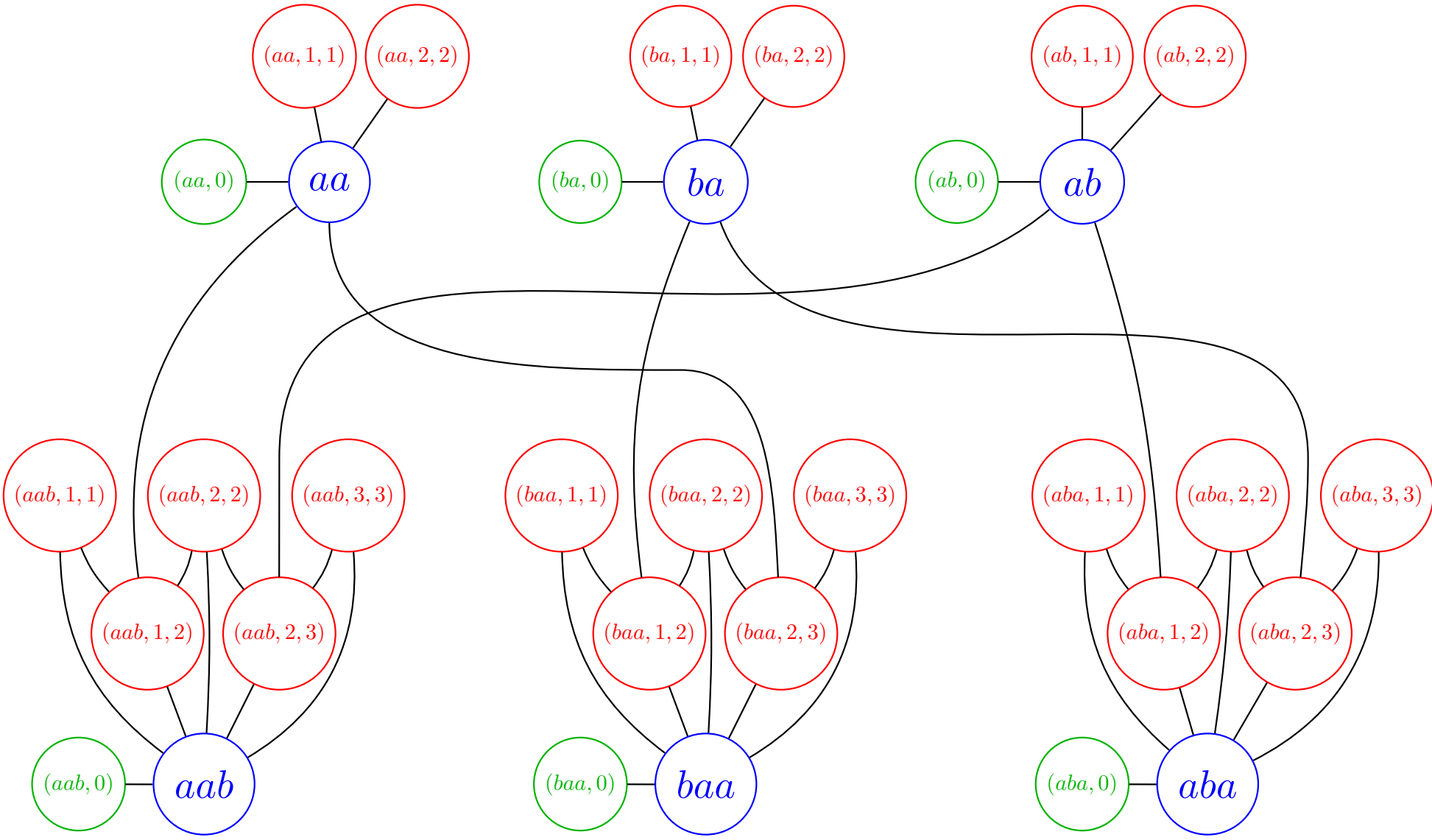
can be computed in polynomial time.

Bounded Number of Non-Terminals

General idea:

- Transform word w into an undirected graph G_w, such that
- independent dominating sets of G_w correspond to grammars for w.
- "Compute minimal independent dominating sets of G_w".

Bounded Number of Non-Terminals



Exact Exponential-Time Algorithms

- Previous approach: $\mathcal{O}(2^{|w|^2})$.
- Enumerating all ordered trees with $|w|$ leaves: $\mathcal{O}(8^{|w|})$.

Exact Exponential-Time Algorithms

- Previous approach: $\mathcal{O}(2^{|w|^2})$.
- Enumerating all ordered trees with $|w|$ leaves: $\mathcal{O}(8^{|w|})$.

Theorem

Smallest 1-level grammars can be computed in time $\mathcal{O}^*(1.8392^{|w|})$.

Proof sketch: Enumerate all factorisations of w without consecutive factors of length 1.

Exact Exponential-Time Algorithms

Obvious dynamic programming approach for multi-level:

- Compute size of best k-level grammar,
- store "last level" (kth level??),
- compute size of best $k+1$-level grammar by somehow extending the last level,
- again store "last level",
- ....

Exact Exponential-Time Algorithms

Obvious dynamic programming approach for multi-level:

- Compute size of best k-level grammar,
- store "last level" (kth level??),
- compute size of best $k+1$-level grammar by somehow extending the last level,
- again store "last level",
- ....

Problem: This does somehow not really work!

Exact Exponential-Time Algorithms

Obvious dynamic programming approach for multi-level:

- Compute size of best k-level grammar,
- store "last level" (kth level??),
- compute size of best $k+1$-level grammar by somehow extending the last level,
- again store "last level",
- ....

Problem: This does somehow not really work!

Solution: Use more sophisticated definition of "levels" (defined somehow by distance from leaves rather than from root).

Exact Exponential-Time Algorithms

Obvious dynamic programming approach for multi-level:

- Compute size of best k-level grammar,
- store "last level" (kth level??),
- compute size of best $k+1$-level grammar by somehow extending the last level,
- again store "last level",
- ....

Problem: This does somehow not really work!

Solution: Use more sophisticated definition of "levels" (defined somehow by distance from leaves rather than from root).

Theorem

Smallest grammars can be computed in time $\mathcal{O}^*(3^{|w|})$.

Thank you very much for your attention