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

ICALP 2016

Context-Free Grammars

$G = (N, \Sigma, R, S)$,

N	set of *nonterminals*,
Σ	*terminal alphabet*,
$S \in N$	*start symbol*,
$R \subseteq N \times (N \cup \Sigma)^+$	set of rules *rules*.

Context-Free Grammars

$G = (N, \Sigma, R, S)$,

$$
\begin{array}{rl}
N & \text{set of } \textit{nonterminals}, \\
\Sigma & \textit{terminal alphabet}, \\
S \in N & \textit{start symbol}, \\
R \subseteq N \times (N \cup \Sigma)^+ & \text{set of rules } \textit{rules}.
\end{array}
$$

G "straight-line program" (or simply grammar) $\Leftrightarrow |L(G)| = 1$

Context-Free Grammars

$G = (N, \Sigma, R, S)$,

$$
\begin{array}{rl}
N & \text{set of } \textit{nonterminals}, \\
\Sigma & \textit{terminal alphabet}, \\
S \in N & \textit{start symbol}, \\
R \subseteq N \times (N \cup \Sigma)^+ & \text{set of rules } \textit{rules}.
\end{array}
$$

G "straight-line program" (or simply grammar) $\Leftrightarrow |L(G)| = 1$

$|G| = \sum_{A \to \alpha \in R} |\alpha|$

Grammar-Based Compression

General idea

Input: word w,
Output: grammar for w.

Grammar-Based Compression

General idea

Input: word w,
Output: grammar for w.

We may ask for

- a shortest grammar.
- a short grammar (but computed fast).
- a grammar that is shortest (short) among all grammars with
  - only k non-terminals (i. e., only k rules),
  - a derivations tree with at most k levels,
  - rules that have right sides of size at most k,
  - ...

Algorithmics on Compressed Strings

Requirement for compression schemes:

- linear or near linear time compression and decompression,
- suitability for solving problems directly on the compressed data.
  $\Rightarrow$ Algorithmics on compressed strings

Algorithmics on Compressed Strings

Requirement for compression schemes:

- linear or near linear time compression and decompression,
- suitability for solving problems directly on the compressed data.
  $\Rightarrow$ Algorithmics on compressed strings

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.

Algorithmics on Compressed Strings

Requirement for compression schemes:

- linear or near linear time compression and decompression,
- suitability for solving problems directly on the compressed data.
  $\Rightarrow$ Algorithmics on compressed strings

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.

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$10\,100\,1000\,10000\,100000\,1000000\,10000000\,100000000\ \ldots$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

10 100 1000 10000 100000 1000000 10000000 100000000 ...

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

A_1 100 1000 10000 100000 1000000 10000000 100000000 $\ldots$
$A_1 \rightarrow 10$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

A_1 100 1000 10000 100000 1000000 10000000 100000000 $\ldots$
$A_1 \rightarrow 10$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\ A_2\ 1000\ 10000\ 100000\ 1000000\ 10000000\ 100000000\ \ldots$

$A_1 \rightarrow 10$

$A_2 \rightarrow A_1 0$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\, A_2\, 1000\, 10000\, 100000\, 1000000\, 10000000\, 100000000\, \ldots$
$A_1 \rightarrow 10$
$A_2 \rightarrow A_1 0$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\ A_2\ A_3\ 10000\ 100000\ 1000000\ 10000000\ 100000000\ \ldots$

$A_1 \rightarrow 10$

$A_2 \rightarrow A_1 0$

$A_3 \rightarrow A_2 0$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\ A_2\ A_3\ 10000\ 100000\ 1000000\ 10000000\ 100000000\ \ldots$

$A_1 \rightarrow 10$

$A_2 \rightarrow A_1 0$

$A_3 \rightarrow A_2 0$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\ A_2\ A_3\ A_4\ 100000\ 1000000\ 10000000\ 100000000\ \ldots$

$A_1 \to 10$

$A_2 \to A_1 0$

$A_3 \to A_2 0$

$A_4 \to A_3 0$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\ A_2\ A_3\ A_4\ 100000\ 1000000\ 10000000\ 100000000\ \ldots$
$A_1 \to 10$
$A_2 \to A_10$
$A_3 \to A_20$
$A_4 \to A_30$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\, A_2\, A_3\, A_4\, A_5\, 1000000\, 10000000\, 100000000\ \ldots$

$A_1 \rightarrow 10$

$A_2 \rightarrow A_1 0$

$A_3 \rightarrow A_2 0$

$A_4 \rightarrow A_3 0$

$A_5 \rightarrow A_4 0$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\, A_2\, A_3\, A_4\, A_5\, 1000000\, 10000000\, 100000000\, \ldots$

$A_1 \rightarrow 10$

$A_2 \rightarrow A_1 0$

$A_3 \rightarrow A_2 0$

$A_4 \rightarrow A_3 0$

$A_5 \rightarrow A_4 0$

$\ldots$

Examples (1/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$A_1\, A_2\, A_3\, A_4\, A_5\, 1000000\, 10000000\, 100000000\, \ldots$
$A_1 \to 10$
$A_2 \to A_1 0$
$A_3 \to A_2 0$
$A_4 \to A_3 0$
$A_5 \to A_4 0$
$\ldots$

$\Rightarrow$ grammar G with $|G| = 3n - 1$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$10\,100\,1000\,10000\,100000\,1000000\,10000000\,100000000\ \ldots$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

1 0 100 1000 10000 100000 1000000 10000000 100000000 ...

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$1{\color{red}A_1}{\color{red}A_1}00\,10000\,100000\,1000000\,10000000\,100000000\ \ldots$

${\color{red}A_1 \to 010}$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$1 A_1 A_1 00\ 10000\ 100000\ 1000000\ 10000000\ 100000000\ \ldots$
$A_1 \to 010$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$1A_1A_1A_2A_2000\,1000000\,10000000\,100000000\ \ldots$
$A_1 \rightarrow 010$
$A_2 \rightarrow 0A_10$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$1A_1A_1A_2A_2\textcolor{red}{000\,1000000\,1000}0000\,100000000\ \ldots$
$A_1 \rightarrow 010$
$A_2 \rightarrow 0A_10$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$1 A_1 A_1 A_2 A_2 A_3 A_3 0000\, 100000000\ \ldots$
$A_1 \rightarrow 010$
$A_2 \rightarrow 0 A_1 0$
$A_3 \rightarrow 0 A_2 0$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$1A_1A_1A_2A_2A_3A_3 0000\,100000000\ \ldots$
$A_1 \to 010$
$A_2 \to 0A_10$
$A_3 \to 0A_20$
$\ldots$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$1A_1A_1A_2A_2A_3A_3 0000\, 100000000\, \ldots$
$A_1 \to 010$
$A_2 \to 0A_10$
$A_3 \to 0A_20$
$\ldots$

$\Rightarrow$ grammar G with $|G| = \frac{5n}{2} + 2k - 3$

Examples (2/2)

$w = \prod_{i=1}^{n} 10^i$, with $n = 2^k$.

$1 A_1 A_1 A_2 A_2 A_3 A_3 0000\, 100000000 \ldots$
$A_1 \to 010$
$A_2 \to 0 A_1 0$
$A_3 \to 0 A_2 0$
$\ldots$

$\Rightarrow$ grammar G with $|G| = \frac{5n}{2} + 2k - 3$

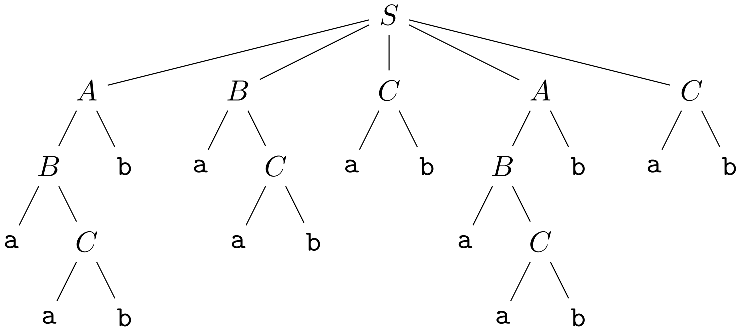
Best grammar we found: $|G| = \frac{9n}{4} + 2k - 2$

Derivation Trees

$$S \to ABCAC\,,$$
$$A \to B\mathtt{b}\,,$$
$$B \to \mathtt{a}C\,,$$
$$C \to \mathtt{ab}\,.$$

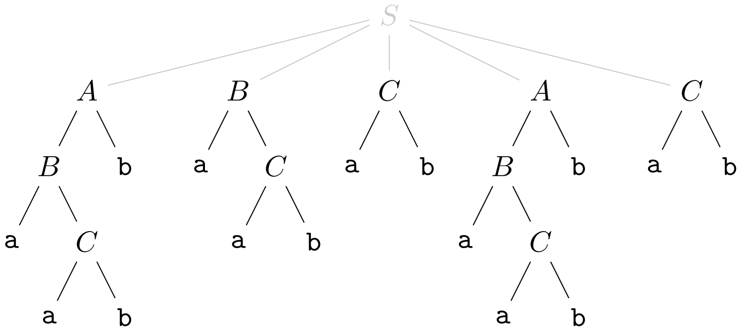
Derivation Trees

$$
\begin{aligned}
S &\to ABCAC\,,\\
A &\to B\texttt{b}\,,\\
B &\to \texttt{a}C\,,\\
C &\to \texttt{ab}\,.
\end{aligned}
$$



Derivation Trees

$$
\begin{aligned}
S &\rightarrow ABCAC\,, \\
A &\rightarrow B\texttt{b}\,, \\
B &\rightarrow \texttt{a}C\,, \\
C &\rightarrow \texttt{ab}\,.
\end{aligned}
$$



1-Level Grammars

$$
\begin{aligned}
S &\to ABCAC\,,\\
A &\to \texttt{aabb}\,,\\
B &\to \texttt{aab}\,,\\
C &\to \texttt{ab}\,.
\end{aligned}
$$

1-Level Grammars

$$\begin{aligned} S &\to ABCAC\,, \\ A &\to \texttt{aabb}\,, \\ B &\to \texttt{aab}\,, \\ C &\to \texttt{ab}\,. \end{aligned}$$

$$\begin{array}{ccccc} A & B & C & A & C \\ | & | & | & | & | \\ \texttt{aabb} & \texttt{aab} & \texttt{ab} & \texttt{aabb} & \texttt{ab} \end{array}$$

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 $|G| \leq k$?

Known Complexity Results

- NP-complete (for unbounded alphabets).

Known Complexity Results

- NP-complete (for unbounded alphabets).
- Many simple and fast approximation algorithms exist.

Known Complexity Results

- NP-complete (for unbounded alphabets).
- Many simple and fast approximation algorithms exist.
- Best known approximation ratio is $\mathcal{O}(\log(\frac{|w|}{m^*}))$ (where m^* is the size of a smallest grammar).

Known Complexity Results

- NP-complete (for unbounded alphabets).
- Many simple and fast approximation algorithms exist.
- Best known approximation ratio is $\mathcal{O}(\log(\frac{|w|}{m^*}))$ (where m^* is the size of a smallest grammar).
- No $\frac{8569}{8568} \approx 1.0001$ approximation ratio (assuming $\mathrm{P} \neq \mathsf{NP}$ and for unbounded alphabets).

Known Complexity Results

- NP-complete (for unbounded alphabets).
- Many simple and fast approximation algorithms exist.
- Best known approximation ratio is $\mathcal{O}(\log(\frac{|w|}{m^*}))$ (where m^* is the size of a smallest grammar).
- No $\frac{8569}{8568} \approx 1.0001$ approximation ratio (assuming $\mathrm{P} \neq \mathsf{NP}$ and for unbounded alphabets).

$\Rightarrow$

Solid theoretical foundation for approximations (or heuristics) missing.

NP-Completeness of the Shortest Grammar Problem

Theorem

1-SGP *is* NP-*complete, even for alphabets of size* 5.

NP-Completeness of the Shortest Grammar Problem

Theorem

1-SGP *is* NP-*complete, even for alphabets of size* 5.

Theorem

SGP *is* NP-*complete, even for alphabets of size* 24.

Proof Ideas

- Reduction from vertex cover (unbounded alphabets):
  - represent edges (v_i, v_j) as factors $\#v_i\#v_j\#$,
  - make sure that only $\#v_i$, $v_i\#$ or $\#v_i\#$ are compressed,
  - grammar is smallest if every $\#v_i\#v_j\#$ is compressed by a $A_i \rightarrow \#v_i\#$ or $A_j \rightarrow \#v_j\#$.

Proof Ideas

- Reduction from vertex cover (unbounded alphabets):
  - represent edges (v_i, v_j) as factors $\#v_i\#v_j\#$,
  - make sure that only $\#v_i$, $v_i\#$ or $\#v_i\#$ are compressed,
  - grammar is smallest if every $\#v_i\#v_j\#$ is compressed by a $A_i \to \#v_i\#$ or $A_j \to \#v_j\#$.
- Finite alphabet: Using symbols or constant size factors for representing vertices (edges) or as separators is not possible ($\Rightarrow$ some encoding needed!).

Proof Ideas

- Reduction from vertex cover (unbounded alphabets):
  - represent edges (v_i, v_j) as factors $\#v_i\#v_j\#$,
  - make sure that only $\#v_i$, $v_i\#$ or $\#v_i\#$ are compressed,
  - grammar is smallest if every $\#v_i\#v_j\#$ is compressed by a $A_i \to \#v_i\#$ or $A_j \to \#v_j\#$.
- Finite alphabet: Using symbols or constant size factors for representing vertices (edges) or as separators is not possible ($\Rightarrow$ some encoding needed!).
- 1-level case: Using unary sequences as separators works! (in 1-level case, separators fully determine the compressed factors.)

Proof Ideas

- Reduction from vertex cover (unbounded alphabets):
  - represent edges (v_i, v_j) as factors $\#v_i\#v_j\#$,
  - make sure that only $\#v_i$, $v_i\#$ or $\#v_i\#$ are compressed,
  - grammar is smallest if every $\#v_i\#v_j\#$ is compressed by a $A_i \to \#v_i\#$ or $A_j \to \#v_j\#$.
- Finite alphabet: Using symbols or constant size factors for representing vertices (edges) or as separators is not possible ($\Rightarrow$ some encoding needed!).
- 1-level case: Using unary sequences as separators works! (in 1-level case, separators fully determine the compressed factors.)
- Multi-level case: We do not know how the encodings will be compressed by a shortest grammar (recall introductory example $10100100010000\ldots$).

Proof Ideas Multi-Level Case

- palindromic codewords: $u \star u^R$, $\star \in \Sigma$, u is 7-ary number.

Proof Ideas Multi-Level Case

- palindromic codewords: $u \star u^R$, $\star \in \Sigma$, u is 7-ary number.
- Crucial properties:
  - overlapping between neighbouring codewords are not repeated $\Rightarrow$ codewords are compressed individually.
  - codewords are produced best "from the middle": $A \to a \star a$, $B \to bAb$, $C \to cBc$, $\ldots$.

Reduction

Graph $G \Rightarrow$ word uvw

Reduction

Graph $G \Rightarrow$ word uvw

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

$$\begin{aligned} \mathbf{v} = & \prod_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} (\# \ \langle \mathbf{7i} + \mathbf{C_v}(\mathbf{i}) \rangle_{\mathbf{v}} \ ¢_{\mathbf{1}} \ \langle \mathbf{7i} - \mathbf{1} \rangle_{\diamond}) \ \$_{\mathbf{2}} \prod_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} (\# \ \langle \mathbf{7i} + \mathbf{C_v}(\mathbf{i}) \rangle_{\mathbf{v}} \ ¢_{\mathbf{2}} \ \langle \mathbf{7i} - \mathbf{2} \rangle_{\diamond}) \ \$_{\mathbf{3}} \\ & \prod_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} (\langle \mathbf{7i} + \mathbf{C_v}(\mathbf{i}) \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7i} - \mathbf{2} \rangle_{\diamond} \ ¢_{\mathbf{1}}) \ \$_{\mathbf{4}} \prod_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} (\langle \mathbf{7i} + \mathbf{C_v}(\mathbf{i}) \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7i} - \mathbf{1} \rangle_{\diamond} \ ¢_{\mathbf{2}}) \ \$_{\mathbf{5}} \\ & \prod_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} (\# \ \langle \mathbf{7i} + \mathbf{C_v}(\mathbf{i}) \rangle_{\mathbf{v}} \ \# \ \langle \mathbf{7i} \rangle_{\diamond}) \ \$_{\mathbf{6}} \end{aligned}$$

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

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:

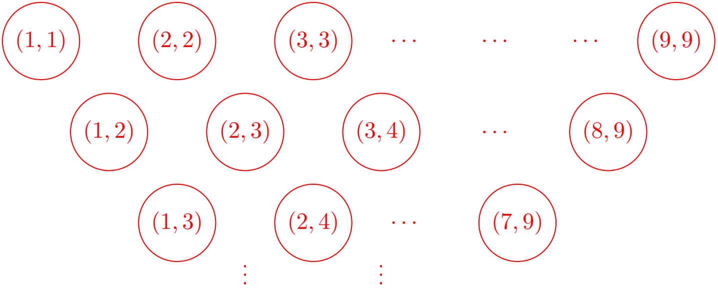
- Transform word w into an undirected graph G_w, such that
- independent dominating sets of G_w correspond to grammars for w.
- "Compute minimal independent dominating sets of G_w".

Bounded Number of Non-Terminals 1-Level Case

$$w = abbababab$$

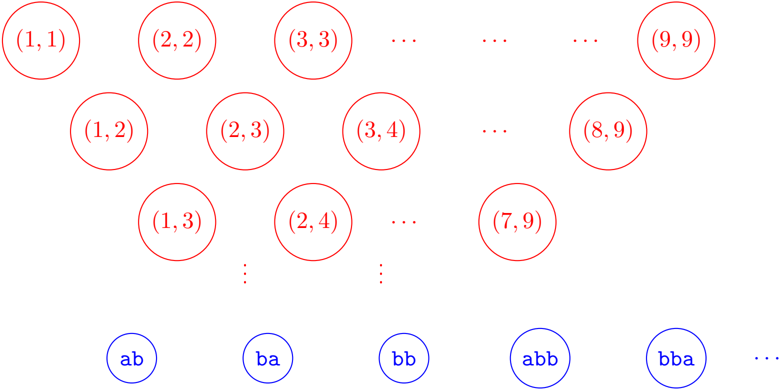
Bounded Number of Non-Terminals 1-Level Case

$$w = abbababab$$



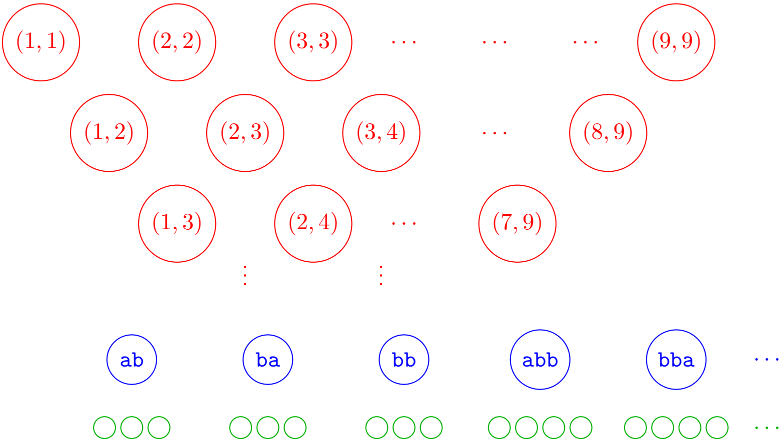
Bounded Number of Non-Terminals 1-Level Case

$$w = abbababab$$



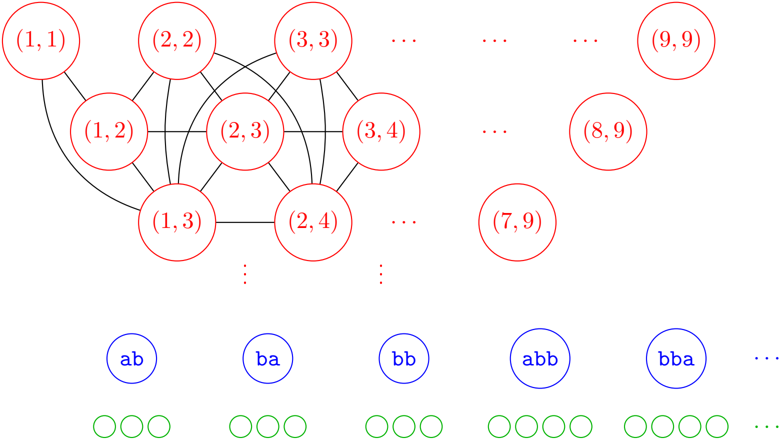
Bounded Number of Non-Terminals 1-Level Case

$$w = abbababab$$



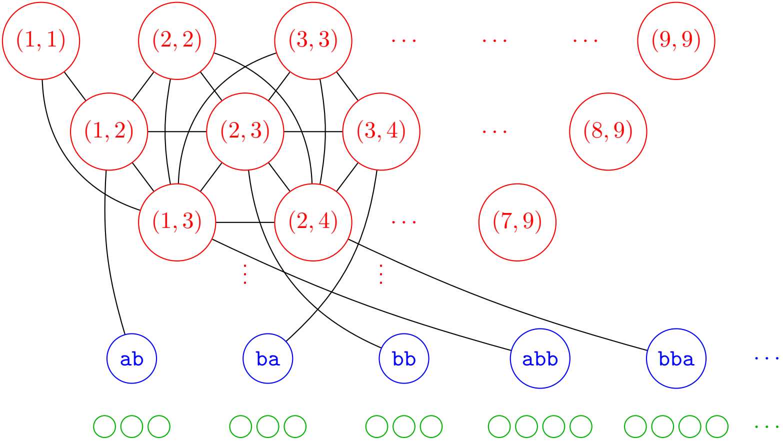
Bounded Number of Non-Terminals 1-Level Case

$$w = abbababab$$



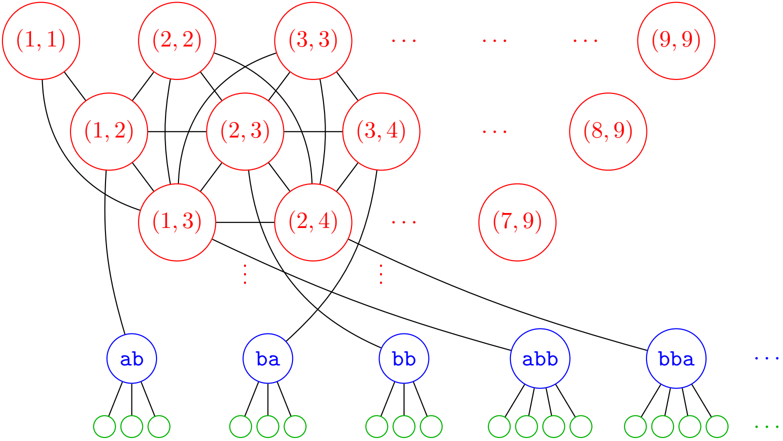
Bounded Number of Non-Terminals 1-Level Case

$$w = abbababab$$

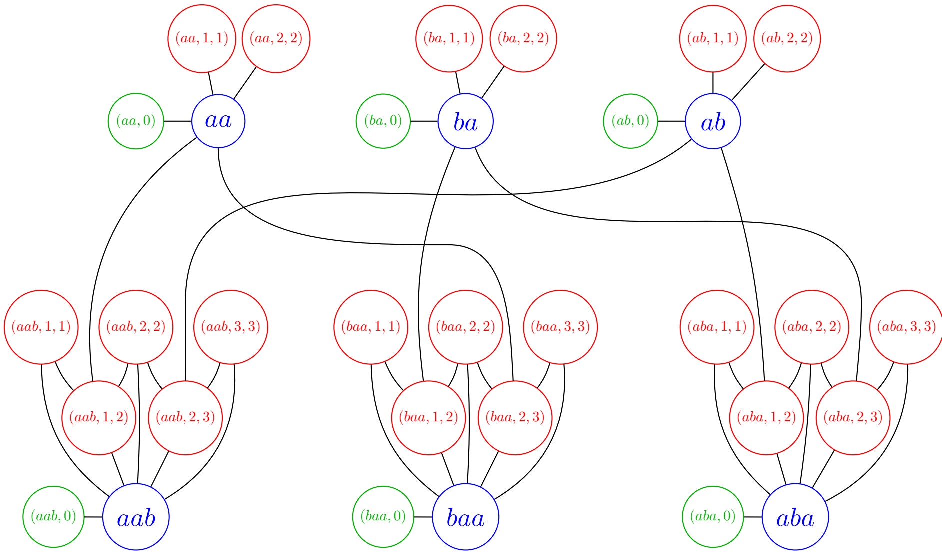


Bounded Number of Non-Terminals 1-Level Case

$$w = abbababab$$



Bounded Number of Non-Terminals Multi-Level Case



Bounded Number of Non-Terminals Multi-Level Case

Theorem

Let $w \in \Sigma^*$ and $k \in \mathbb{N}$. A 1-level grammar for w with at most k rules that is minimal among all 1-level grammars for w with at most k rules can be computed in time $\mathcal{O}(|w|^{2k+4})$.

Theorem

Let $w \in \Sigma^*$ and $k \in \mathbb{N}$. A grammar for w with at most k rules that is minimal among all grammars for w with at most k rules can be computed in time $\mathcal{O}(|w|^{2k+6})$.

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",
- $\ldots$.

Exact Exponential-Time Algorithms

`a a a b b a b a a a a a a b b a b`

Exact Exponential-Time Algorithms

a a a b b a b a a a a a a b b a b

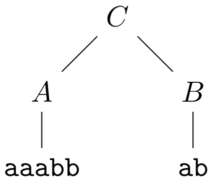
Exact Exponential-Time Algorithms

A	B	`aaa`	A	B
`aaabb`	`ab`		`aaabb`	`ab`

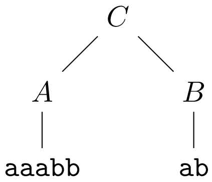
Exact Exponential-Time Algorithms

$$\begin{array}{ccccc} A & B & \texttt{aaa} & A & B \\ | & | & & | & | \\ \texttt{aaabb} & \texttt{ab} & & \texttt{aaabb} & \texttt{ab} \end{array}$$

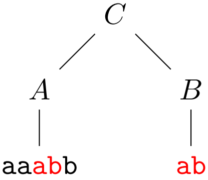
Exact Exponential-Time Algorithms



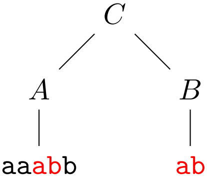
`aaa`



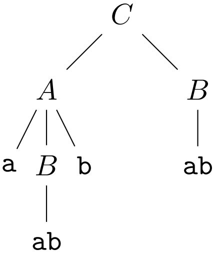
Exact Exponential-Time Algorithms



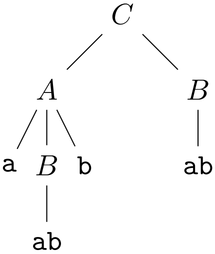
aaa



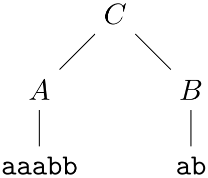
Exact Exponential-Time Algorithms



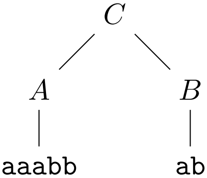
aaa



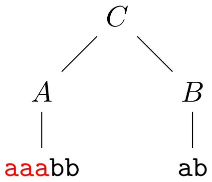
Exact Exponential-Time Algorithms



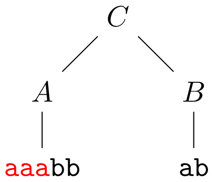
aaa



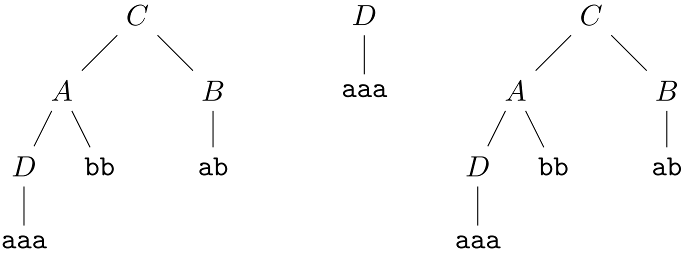
Exact Exponential-Time Algorithms



aaa



Exact Exponential-Time Algorithms



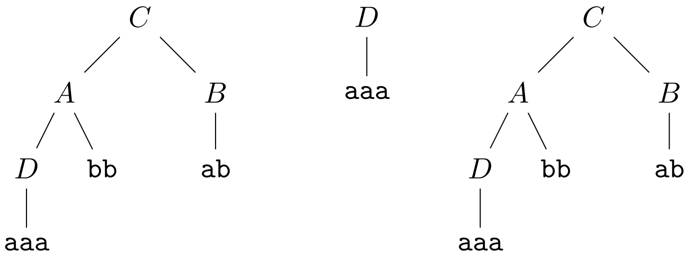
Exact Exponential-Time Algorithms

Solution: More sophisticated definition of “levels”.
$N_i = \{A \in N \mid A \text{ yields terminal string in } i \text{ steps}\}$.

Exact Exponential-Time Algorithms

Solution: More sophisticated definition of "levels".

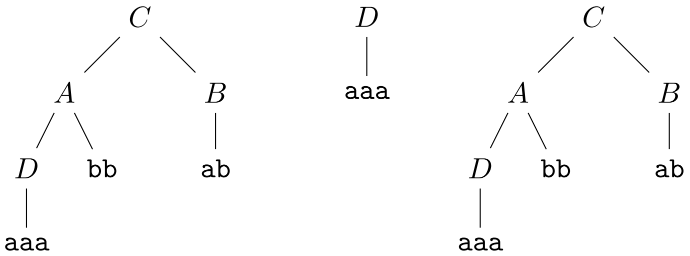
$N_i = \{A \in N \mid A \text{ yields terminal string in } i \text{ steps}\}$.



Exact Exponential-Time Algorithms

Solution: More sophisticated definition of "levels".

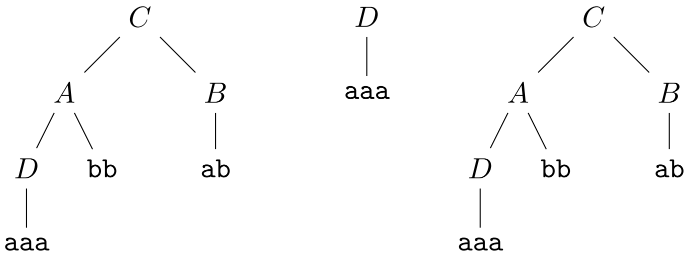
$N_i = \{A \in N \mid A \text{ yields terminal string in } i \text{ steps}\}$.



$N_1 = \{B, D\}$, $N_2 = \{A\}$, $N_3 = \{C\}$,

Exact Exponential-Time Algorithms

Solution: More sophisticated definition of “levels”.
$N_i = \{A \in N \mid A \text{ yields terminal string in } i \text{ steps}\}$.

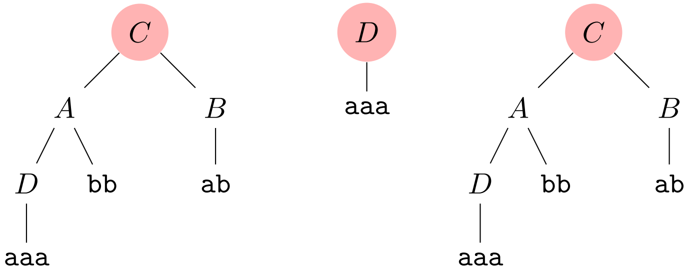


$N_1 = \{B, D\}$, $N_2 = \{A\}$, $N_3 = \{C\}$,

1[st] level: compressed string,
2[nd] level: derive all N_k in 1[st] level,
3[rd] level: derive all N_{k-1} in 2[nd] level,
$\dots$

Exact Exponential-Time Algorithms

Solution: More sophisticated definition of “levels”.
$N_i = \{A \in N \mid A \text{ yields terminal string in } i \text{ steps}\}$.

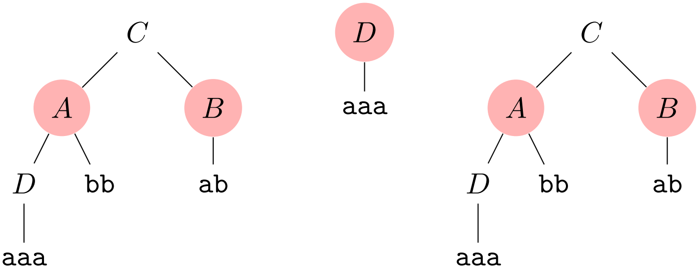


$N_1 = \{B, D\}$, $N_2 = \{A\}$, $N_3 = \{C\}$,

1[st] level: compressed string, CDC
2[nd] level: derive all N_k in 1[st] level,
3[rd] level: derive all N_{k-1} in 2[nd] level,
…

Exact Exponential-Time Algorithms

Solution: More sophisticated definition of “levels”.
$N_i = \{A \in N \mid A \text{ yields terminal string in } i \text{ steps}\}$.

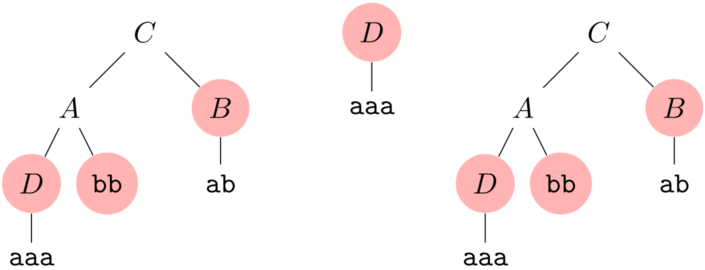


$N_1 = \{B, D\}$, $N_2 = \{A\}$, $N_3 = \{C\}$,

1st level: compressed string, CDC
2nd level: derive all N_k in 1st level, $ABDAB$
3rd level: derive all N_{k-1} in 2nd level,
...

Exact Exponential-Time Algorithms

Solution: More sophisticated definition of "levels".
$N_i = \{A \in N \mid A \text{ yields terminal string in } i \text{ steps}\}$.



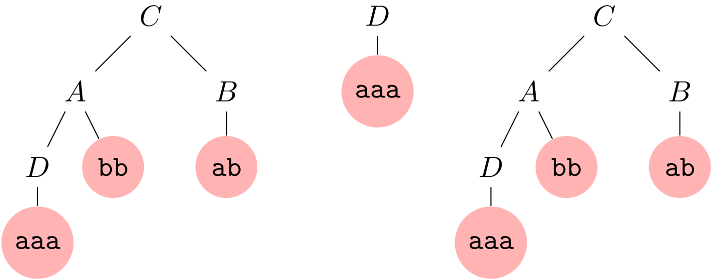
$N_1 = \{B, D\}$, $N_2 = \{A\}$, $N_3 = \{C\}$,

1st level: compressed string, CDC
2nd level: derive all N_k in 1st level, $ABDAB$
3rd level: derive all N_{k-1} in 2nd level, $D\texttt{bb}BDD\texttt{bb}B$
...

Exact Exponential-Time Algorithms

Solution: More sophisticated definition of "levels".

$N_i = \{A \in N \mid A \text{ yields terminal string in } i \text{ steps}\}$.



$N_1 = \{B, D\}$, $N_2 = \{A\}$, $N_3 = \{C\}$,

1st level: compressed string, CDC

2nd level: derive all N_k in 1st level, $ABDAB$

3rd level: derive all N_{k-1} in 2nd level, $D\texttt{bb}BDD\texttt{bb}B$

... `aaabbabaaaaabbab`

Dynamic Programming Algorithm

k^{th} $AB\texttt{a}C\texttt{b}CB\texttt{a}$ $w_1w_2\ldots w_8$

Dynamic Programming Algorithm

k^{th}	$AB\texttt{a}C\texttt{b}CB\texttt{a}$	$w_1 w_2 \ldots w_8$
$(k+1)^{\text{th}}$	$AB\texttt{a}D\texttt{b}A\texttt{b}D\texttt{b}AB\texttt{a}$	$w_1 \ldots w_{4,1} w_{4,2} w_{4,3} \ldots w_{6,1} w_{6,2} w_{6,3} \ldots w_8$

Dynamic Programming Algorithm

k^{th}	$AB\mathtt{a}C\mathtt{b}CB\mathtt{a}$	$w_1 w_2 \ldots w_8$
$(k+1)^{\text{th}}$	$AB\mathtt{a}D\mathtt{b}A\mathtt{b}D\mathtt{b}AB\mathtt{a}$	$w_1 \ldots w_{4,1} w_{4,2} w_{4,3} \ldots w_{6,1} w_{6,2} w_{6,3} \ldots w_8$

Dynamic Programming Algorithm

k^{th}	$AB\texttt{a}C\texttt{b}CB\texttt{a}$	$w_1 w_2 \ldots w_8$
$(k+1)^{\text{th}}$	$AB\texttt{a}D\texttt{b}A\texttt{b}D\texttt{b}AB\texttt{a}$	$w_1 \ldots w_{4,1} w_{4,2} w_{4,3} \ldots w_{6,1} w_{6,2} w_{6,3} \ldots w_8$

Sufficient local information:

Size of smallest k-level grammar

$$\begin{array}{ccccccc} u & \ldots & v & \ldots & u & \ldots & v \\ (u_1 \ldots u_m) & \ldots & (v_1 \ldots v_\ell) & \ldots & (u_1 \ldots u_m) & \ldots & (v_1 \ldots v_\ell) \end{array}$$

Dynamic Programming Algorithm

k^{th}	$AB\mathtt{a}C\mathtt{b}CB\mathtt{a}$	$w_1 w_2 \dots w_8$
$(k+1)^{\text{th}}$	$AB\mathtt{a}D\mathtt{b}A\mathtt{b}D\mathtt{b}AB\mathtt{a}$	$w_1 \dots w_{4,1} w_{4,2} w_{4,3} \dots w_{6,1} w_{6,2} w_{6,3} \dots w_8$

Sufficient local information:

Size of smallest k-level grammar

$$\begin{array}{ccccccc} u & \dots & v & \dots & u & \dots & v \\ (u_1 \dots u_m) & \dots & (v_1 \dots v_\ell) & \dots & (u_1 \dots u_m) & \dots & (v_1 \dots v_\ell) \end{array}$$

Theorem

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

Thank you very much for your attention