Compressed Membership in Automata with Compressed Labels

Markus Lohrey
Christian Mathissen

University of Leipzig

June 16, 2011

Motivation

Try to develop algorithms that directly work on compressed data.

Motivation

Try to develop algorithms that directly work on compressed data.

Goal: Beat straightforward decompress and manipulate strategy.

Motivation

Try to develop algorithms that directly work on compressed data.

Goal: Beat straightforward decompress and manipulate strategy.

In this talk: focus on compressed strings

Motivation

Try to develop algorithms that directly work on compressed data.

Goal: Beat straightforward decompress and manipulate strategy.

In this talk: focus on compressed strings

Applications:

- all domains, where massive string data arise and have to be processed, e.g. bioinformatics

Motivation

Try to develop algorithms that directly work on compressed data.

Goal: Beat straightforward decompress and manipulate strategy.

In this talk: focus on compressed strings

Applications:

- all domains, where massive string data arise and have to be processed, e.g. bioinformatics
- large (and highly compressible) strings often occur as intermediate data structures (e.g. in computational group theory, program analysis, verification).

Straight-line programs

Dictionary-based compression algorithms (LZ77, LZ78) exploit text repetition.

Straight-line programs

Dictionary-based compression algorithms (LZ77, LZ78) exploit text repetition.

Straight-line programs are a general representation for compressed strings, which covers most dictionary-based algorithms.

Straight-line programs

Dictionary-based compression algorithms (LZ77, LZ78) exploit text repetition.

Straight-line programs are a general representation for compressed strings, which covers most dictionary-based algorithms.

A straight-line program (SLP) over the alphabet Γ is a sequence of definitions $\mathbb{A} = (A_i := \alpha_i)_{1 \leq i \leq n}$, where either $\alpha_i \in \Gamma$ or $\alpha_i = A_j A_k$ for some $j, k < i$.

Straight-line programs

Dictionary-based compression algorithms (LZ77, LZ78) exploit text repetition.

Straight-line programs are a general representation for compressed strings, which covers most dictionary-based algorithms.

A straight-line program (SLP) over the alphabet Γ is a sequence of definitions $\mathbb{A} = (A_i := \alpha_i)_{1 \leq i \leq n}$, where either $\alpha_i \in \Gamma$ or $\alpha_i = A_j A_k$ for some $j, k < i$.

Alternatively: a context-free grammar that generates exactly one string.

Straight-line programs

Dictionary-based compression algorithms (LZ77, LZ78) exploit text repetition.

Straight-line programs are a general representation for compressed strings, which covers most dictionary-based algorithms.

A straight-line program (SLP) over the alphabet Γ is a sequence of definitions $\mathbb{A} = (A_i := \alpha_i)_{1 \leq i \leq n}$, where either $\alpha_i \in \Gamma$ or $\alpha_i = A_j A_k$ for some $j, k < i$.

Alternatively: a context-free grammar that generates exactly one string.

We write val($\mathbb{A}$) for the unique word generated by $\mathbb{A}$.

Straight-line programs

Dictionary-based compression algorithms (LZ77, LZ78) exploit text repetition.

Straight-line programs are a general representation for compressed strings, which covers most dictionary-based algorithms.

A straight-line program (SLP) over the alphabet Γ is a sequence of definitions $\mathbb{A} = (A_i := \alpha_i)_{1 \le i \le n}$, where either $\alpha_i \in \Gamma$ or $\alpha_i = A_j A_k$ for some $j, k < i$.

Alternatively: a context-free grammar that generates exactly one string.

We write $\mathsf{val}(\mathbb{A})$ for the unique word generated by $\mathbb{A}$.

The size of an SLP $\mathbb{A} = (A_i := \alpha_i)_{1 \le i \le n}$ is $|\mathbb{A}| = n$.

Straight-line programs

Dictionary-based compression algorithms (LZ77, LZ78) exploit text repetition.

Straight-line programs are a general representation for compressed strings, which covers most dictionary-based algorithms.

A straight-line program (SLP) over the alphabet Γ is a sequence of definitions $\mathbb{A} = (A_i := \alpha_i)_{1 \leq i \leq n}$, where either $\alpha_i \in \Gamma$ or $\alpha_i = A_j A_k$ for some $j, k < i$.

Alternatively: a context-free grammar that generates exactly one string.

We write $\mathrm{val}(\mathbb{A})$ for the unique word generated by $\mathbb{A}$.

The size of an SLP $\mathbb{A} = (A_i := \alpha_i)_{1 \leq i \leq n}$ is $|\mathbb{A}| = n$.

One may have $|\mathrm{val}(\mathbb{A})| = 2^n$.

Straight-line programs

Dictionary-based compression algorithms (LZ77, LZ78) exploit text repetition.

Straight-line programs are a general representation for compressed strings, which covers most dictionary-based algorithms.

A straight-line program (SLP) over the alphabet Γ is a sequence of definitions $\mathbb{A} = (A_i := \alpha_i)_{1 \leq i \leq n}$, where either $\alpha_i \in \Gamma$ or $\alpha_i = A_j A_k$ for some $j, k < i$.

Alternatively: a context-free grammar that generates exactly one string.

We write $\mathrm{val}(\mathbb{A})$ for the unique word generated by $\mathbb{A}$.

The size of an SLP $\mathbb{A} = (A_i := \alpha_i)_{1 \leq i \leq n}$ is $|\mathbb{A}| = n$.

One may have $|\mathrm{val}(\mathbb{A})| = 2^n$.

Thus, an SLP $\mathbb{A}$ can be seen as a compressed representation of $\mathrm{val}(\mathbb{A})$.

Straight-line programs

Example: $\mathbb{A} = (A_1 := b, \quad A_2 := a, \quad A_i := A_{i-1}A_{i-2}$ for $3 \leq i \leq 7)$.

Straight-line programs

Example: $\mathbb{A} = (A_1 := b, \quad A_2 := a, \quad A_i := A_{i-1}A_{i-2}$ for $3 \leq i \leq 7)$.

Then $\mathrm{val}(\mathbb{A}) = abaababaabaab$:

Straight-line programs

Example: $\mathbb{A} = (A_1 := b, \quad A_2 := a, \quad A_i := A_{i-1}A_{i-2}$ for $3 \leq i \leq 7)$.

Then $\mathrm{val}(\mathbb{A}) = abaababaabaab$:

$$
\begin{aligned}
A_3 &= A_2A_1 = ab \\
A_4 &= A_3A_2 = aba \\
A_5 &= A_4A_3 = abaab \\
A_6 &= A_5A_4 = abaababa \\
A_7 &= A_6A_5 = abaababaabaab
\end{aligned}
$$

Straight-line programs

Example: $\mathbb{A} = (A_1 := b, \quad A_2 := a, \quad A_i := A_{i-1}A_{i-2}$ for $3 \leq i \leq 7)$.

Then $\mathrm{val}(\mathbb{A}) = abaababaabaab$:

$$\begin{aligned} A_3 &= A_2 A_1 = ab \\ A_4 &= A_3 A_2 = aba \\ A_5 &= A_4 A_3 = abaab \\ A_6 &= A_5 A_4 = abaababa \\ A_7 &= A_6 A_5 = abaababaabaab \end{aligned}$$

Relationship to dictionary-based compression (Rytter):

Straight-line programs

Example: $\mathbb{A} = (A_1 := b, \quad A_2 := a, \quad A_i := A_{i-1}A_{i-2}$ for $3 \leq i \leq 7)$.

Then $\mathrm{val}(\mathbb{A}) = abaababaabaab$:

$$\begin{aligned} A_3 &= A_2A_1 = ab \\ A_4 &= A_3A_2 = aba \\ A_5 &= A_4A_3 = abaab \\ A_6 &= A_5A_4 = abaababa \\ A_7 &= A_6A_5 = abaababaabaab \end{aligned}$$

Relationship to dictionary-based compression (Rytter):

- From an SLP $\mathbb{A}$ one can compute in polynomial time LZ77($\mathrm{val}(\mathbb{A})$).

Straight-line programs

Example: $\mathbb{A} = (A_1 := b, \quad A_2 := a, \quad A_i := A_{i-1}A_{i-2}$ for $3 \leq i \leq 7)$.
Then $\mathrm{val}(\mathbb{A}) = abaababaabaab$:

$$\begin{aligned} A_3 &= A_2A_1 = ab \\ A_4 &= A_3A_2 = aba \\ A_5 &= A_4A_3 = abaab \\ A_6 &= A_5A_4 = abaababa \\ A_7 &= A_6A_5 = abaababaabaab \end{aligned}$$

Relationship to dictionary-based compression (Rytter):

- From an SLP $\mathbb{A}$ one can compute in polynomial time LZ77($\mathrm{val}(\mathbb{A})$).
- From LZ77(w) one can compute in polynomial time an SLP $\mathbb{A}$ with $\mathrm{val}(\mathbb{A}) = w$.

Algorithms on SLP-compressed strings

The mother of all algorithms on SLP -compressed strings:

Algorithms on SLP-compressed strings

The mother of all algorithms on SLP -compressed strings:

Plandowski 1994

The following problem can be solved in polynomial time:

INPUT: SLPs $\mathbb{A}, \mathbb{B}$
QUESTION: $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$?

Algorithms on SLP-compressed strings

The mother of all algorithms on SLP -compressed strings:

Plandowski 1994

The following problem can be solved in polynomial time:

INPUT: SLPs $\mathbb{A}, \mathbb{B}$
QUESTION: $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$?

Note: The decompress-and-compare strategy does not work here.
We cannot compute $\mathsf{val}(\mathbb{A})$ and $\mathsf{val}(\mathbb{B})$ in polynomial time.

Algorithms on SLP-compressed strings

The mother of all algorithms on SLP -compressed strings:

Plandowski 1994

The following problem can be solved in polynomial time:

INPUT: SLPs $\mathbb{A}, \mathbb{B}$
QUESTION: $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$?

Note: The decompress-and-compare strategy does not work here.
We cannot compute $\mathsf{val}(\mathbb{A})$ and $\mathsf{val}(\mathbb{B})$ in polynomial time.

Plandowski's algorithm uses combinatorics on words, in particular the Periodicity-Lemma of Fine and Wilf.

Improvements of Plandowski's result

Gasieniec, Karpinski, Miyazaki, Plandowski, Rytter, Shinohara, Takeda (mid 90's)

The following problem can be solved in polynomial time
(fully compressed pattern matching):

INPUT: SLPs $\mathbb{P}, \mathbb{T}$
QUESTION: Is val($\mathbb{P}$) a factor of val($\mathbb{T}$), i.e., $\exists x, y : \text{val}(\mathbb{T}) = x \, \text{val}(\mathbb{P}) \, y$?

Improvements of Plandowski's result

Gasieniec, Karpinski, Miyazaki, Plandowski, Rytter, Shinohara, Takeda (mid 90's)

The following problem can be solved in polynomial time (fully compressed pattern matching):

INPUT: SLPs $\mathbb{P}, \mathbb{T}$
QUESTION: Is $\mathrm{val}(\mathbb{P})$ a factor of $\mathrm{val}(\mathbb{T})$, i.e., $\exists x, y : \mathrm{val}(\mathbb{T}) = x \, \mathrm{val}(\mathbb{P}) \, y$?

The best known algorithm has a running time of $O(|\mathbb{P}| \cdot |\mathbb{T}|^2)$ (Lifshits 2006).

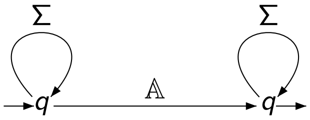
Generalization: Compressed automata

A compressed automata is an ordinary finite automaton, where every transition is labelled with an SLP.

Generalization: Compressed automata

A compressed automata is an ordinary finite automaton, where every transition is labelled with an SLP.

Example: The compressed automaton

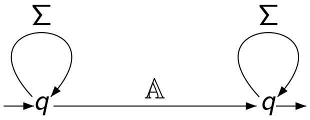


accepts all words that have val($\mathbb{A}$) as a factor.

Generalization: Compressed automata

A compressed automata is an ordinary finite automaton, where every transition is labelled with an SLP.

Example: The compressed automaton



accepts all words that have val($\mathbb{A}$) as a factor.

The size $|\mathcal{A}|$ of the compressed automaton $\mathcal{A}$ (with the set Δ of transition triples) is

$$|\mathcal{A}| = \sum_{(p,\mathbb{A},q)\in\Delta} |\mathbb{A}|.$$

Compressed membership for compressed automata

Compressed membership for compressed automata:

INPUT: A compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$.
QUESTION: $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$?

Compressed membership for compressed automata

Compressed membership for compressed automata:

INPUT: A compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$.
QUESTION: $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$?

Plandowski, Rytter 1999

- Compressed membership for compressed automata belongs to PSPACE.
- Compressed membership for compressed automata over a unary alphabet is NP-complete.

Compressed membership for compressed automata

Compressed membership for compressed automata:

INPUT: A compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$.
QUESTION: $\mathrm{val}(\mathbb{B}) \in L(\mathcal{A})$?

Plandowski, Rytter 1999

- Compressed membership for compressed automata belongs to PSPACE.
- Compressed membership for compressed automata over a unary alphabet is NP-complete.

Plandowski and Rytter conjectured that compressed membership for compressed automata is NP-complete (for every alphabet size).

Some combinatorics on words

The period of the word $w \in \Sigma^*$ is the smallest number p such that
$w[k + p] = w[k]$ for all $1 \leq k \leq |w| - p$
(period(w) $= |w|$ if such a p does not exist).

Some combinatorics on words

The period of the word $w \in \Sigma^*$ is the smallest number p such that $w[k+p] = w[k]$ for all $1 \leq k \leq |w| - p$ (period$(w) = |w|$ if such a p does not exist).

Let order$(w) = \lfloor \frac{|w|}{\text{period}(w)} \rfloor$.

Some combinatorics on words

The period of the word $w \in \Sigma^*$ is the smallest number p such that $w[k+p] = w[k]$ for all $1 \leq k \leq |w| - p$
(period$(w) = |w|$ if such a p does not exist).

Let order$(w) = \lfloor \frac{|w|}{\text{period}(w)} \rfloor$.

Example: Let $w = abbabbab = (abb)^2 ab$.

Some combinatorics on words

The period of the word $w \in \Sigma^*$ is the smallest number p such that $w[k+p] = w[k]$ for all $1 \leq k \leq |w| - p$ (period(w) $= |w|$ if such a p does not exist).

Let $\mathrm{order}(w) = \lfloor \frac{|w|}{\mathrm{period}(w)} \rfloor$.

Example: Let $w = abbabbab = (abb)^2 ab$.

Then $\mathrm{period}(w) = 3$ and $\mathrm{order}(w) = 2$.

Some combinatorics on words

The period of the word $w \in \Sigma^*$ is the smallest number p such that $w[k+p] = w[k]$ for all $1 \leq k \leq |w| - p$ ($\text{period}(w) = |w|$ if such a p does not exist).

Let $\text{order}(w) = \lfloor \frac{|w|}{\text{period}(w)} \rfloor$.

Example: Let $w = abbabbab = (abb)^2 ab$.
Then $\text{period}(w) = 3$ and $\text{order}(w) = 2$.

For a compressed automaton $\mathcal{A}$, let:

$$\begin{aligned} \text{order}(\mathcal{A}) &= \max\{\text{order}(\text{val}(\mathbb{A})) \mid \mathbb{A} \text{ occurs as a label in } \mathcal{A}\} \\ \text{period}(\mathcal{A}) &= \max\{\text{period}(\text{val}(\mathbb{A})) \mid \mathbb{A} \text{ occurs as a label in } \mathcal{A}\} \end{aligned}$$

First main result

Theorem 1

For a compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$, we can check $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ in time polynomial in (i) $|\mathcal{A}|$, (ii) $|\mathbb{B}|$, and (iii) $\mathsf{order}(\mathcal{A})$.

First main result

Theorem 1

For a compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$, we can check $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ *in time polynomial in (i)* $|\mathcal{A}|$, *(ii)* $|\mathbb{B}|$, *and (iii)* $\mathsf{order}(\mathcal{A})$.

We use the following combinatorial fact:

Let $u, v \in \Sigma^*$ and let p be a position in v. Then there exist at most $\mathsf{order}(u)$ many occurrences of u in v that "touch" position p.

Second main result

Theorem 2

For a compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$, we can check $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ *nondeterministically in time polynomial in (i)* $|\mathcal{A}|$*, (ii)* $|\mathbb{B}|$*, and (iii)* $\mathsf{period}(\mathcal{A})$*.*

Second main result

Theorem 2

For a compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$, we can check $\mathrm{val}(\mathbb{B}) \in L(\mathcal{A})$ *nondeterministically in time polynomial in (i)* $|\mathcal{A}|$*, (ii)* $|\mathbb{B}|$*, and (iii)* $\mathrm{period}(\mathcal{A})$*.*

Generalizes the result of Plandowski & Rytter for a unary alphabet (NP-completeness of compressed membership for compressed automata).

Second main result

Theorem 2

For a compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$, we can check $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ *nondeterministically in time polynomial in (i)* $|\mathcal{A}|$, *(ii)* $|\mathbb{B}|$, *and (iii)* $\mathsf{period}(\mathcal{A})$.

Generalizes the result of Plandowski & Rytter for a unary alphabet (NP-completeness of compressed membership for compressed automata).

A word $w = a^n$ has period 1!

Second main result

Theorem 2

For a compressed automaton $\mathcal{A}$ and an SLP $\mathbb{B}$, we can check $\text{val}(\mathbb{B}) \in L(\mathcal{A})$ *nondeterministically in time polynomial in (i)* $|\mathcal{A}|$*, (ii)* $|\mathbb{B}|$*, and (iii)* $\text{period}(\mathcal{A})$*.*

Generalizes the result of Plandowski & Rytter for a unary alphabet (NP-completeness of compressed membership for compressed automata).

A word $w = a^n$ has period 1!

The theorem is proven by a reduction to the case of a unary alphabet.

Open problems and a conjecture

Conjecture 1

Compressed membership for compressed automata is NP-complete.

Open problems and a conjecture

Conjecture 1

Compressed membership for compressed automata is NP-complete.

Conjecture 2

If $\mathrm{val}(\mathbb{B}) \in L(\mathcal{A})$ for an SLP $\mathbb{B}$ and a compressed automaton $\mathcal{A}$, then there exists an accepting run of $\mathcal{A}$ on $\mathrm{val}(\mathbb{B})$ (viewed as a word over the set of transition triples of $\mathcal{A}$), which can be generated by an SLP of size $\mathrm{poly}(|\mathbb{B}|, |\mathcal{A}|)$.

Open problems and a conjecture

Conjecture 1

Compressed membership for compressed automata is NP-complete.

Conjecture 2

If $\mathrm{val}(\mathbb{B}) \in L(\mathcal{A})$ for an SLP $\mathbb{B}$ and a compressed automaton $\mathcal{A}$, then there exists an accepting run of $\mathcal{A}$ on $\mathrm{val}(\mathbb{B})$ (viewed as a word over the set of transition triples of $\mathcal{A}$), which can be generated by an SLP of size $\mathrm{poly}(|\mathbb{B}|, |\mathcal{A}|)$.

Conjecture 2 implies Conjecture 1:

- Guess nondeterministically an SLP $\mathbb{C}$ of polynomial size.
- Check (in deterministic polynomial time), whether $\mathrm{val}(\mathbb{C})$ is an accepting run of $\mathcal{A}$ on $\mathrm{val}(\mathbb{B})$.