Word equations in nondeterministic linear space

Artur Jeż
Institute of Computer Science
International Colloquium on Automata, Languages and Programming
Warszawa, 13.07.2017
Definition

Given equation $U = V$, where $U, V \in (\Sigma \cup \mathcal{X})^*$. Is there a substitution $S : \mathcal{X} \rightarrow \Sigma^*$ satisfying the equation?
Definition

Given equation $U = V$, where $U, V \in (\Sigma \cup \mathcal{X})^*$. Is there a substitution $S : \mathcal{X} \rightarrow \Sigma^*$ satisfying the equation?

\[
a \ X \ b \ X \ Y \ bbb = X \ abaab \ Y \ bY \quad S(X) = aa, \ S(Y) = bb
\]
Word Equations

**Definition**

Given equation \( U = V \), where \( U, V \in (\Sigma \cup \mathcal{X})^* \).

Is there a substitution \( S : \mathcal{X} \to \Sigma^* \) satisfying the equation?

\[
\begin{align*}
axbxYbbb &= XabaabYby & S(X) &= aa, S(Y) &= bb \\
aaaabaabb&&=aaaabaabbbbbb
\end{align*}
\]
**Definition**

Given equation $U = V$, where $U, V \in (\Sigma \cup \mathcal{X})^*$. Is there a substitution $S : \mathcal{X} \rightarrow \Sigma^*$ satisfying the equation?

\[
aXbXYbbb = XabaabYbY \quad S(X) = aa, S(Y) = bb
\]
\[
aaabaabbbbbb = aaabaabbbbbb
\]

We extend $S$ to a $S : (\Sigma \cup \mathcal{X})^* \rightarrow \Sigma^*$; identity on $\Sigma$. $S(U)$ is a **solution word**.

**Lenght-minimal $S$**: minimises $|S(U)|$
Definition

Given equation $U = V$, where $U, V \in (\Sigma \cup \mathcal{X})^*$. Is there a substitution $S : \mathcal{X} \to \Sigma^*$ satisfying the equation?

$$a X b X Y bbb = X abaab Y bY \quad S(X) = aa, \quad S(Y) = bb$$

$$aaabaaabbbbb = aaaabaaabbbbbb$$

We extend $S$ to a $S : (\Sigma \cup \mathcal{X})^* \to \Sigma^*$; identity on $\Sigma$.

$S(U)$ is a solution word.

Length-minimal $S$: minimises $|S(U)|$

This is important

- unification
- word combinatorics
- helpful in equations in free group (and other)
Makanin ’77 4NEXPTIME
Makanin ’77 4NEXPTIME

[...]
Makanin ’77 4NEXPTIME

[...]

Gutierrez ’98 EXPSPACE
Makanin ’77 4NEXPTIME

[...]

Gutierrez ’98 EXPSPACE

Plandowski & Rytter ’98 new approach — using compression
Makanin ’77 4NEXPTIME

[...]

Gutierrez ’98 EXPSPACE

Plandowski & Rytter ’98 new approach — using compression

Plandowski ’99 PSPACE
Makanin ’77 4NEXPTIME

[...] 

Gutierrez ’98 EXPSPACE

Plandowski & Rytter ’98 new approach — using compression

Plandowski ’99 PSPACE

J. ’13 PSPACE
Algorithms and Complexity

- Makanin ’77 4NEXPTIME
  
  […]

- Gutierrez ’98 EXPSPACE

- Plandowski & Rytter ’98 new approach — using compression

- Plandowski ’99 PSPACE

  J. ’13 PSPACE

- NP-hard, believed to be in NP
Makanin ’77 4NEXPTIME

[...]

Gutierrez ’98 EXPSPACE

Plandowski & Rytter ’98 new approach — using compression

Plandowski ’99 PSPACE

J. ’13 PSPACE

- NP-hard, believed to be in NP
- Exact space complexity?
Makanin ’77 4NEXPTIME

 […]

Gutierrez ’98 EXPSPACE exponential

Plandowski & Rytter ’98 new approach — using compression

Plandowski ’99 PSPACE $O(n^5)$

J. ’13 PSPACE $O(n \log n)$

▶ NP-hard, believed to be in NP

▶ Exact space complexity?
Makanin ’77 4NEXPTIME

[...]

Gutierrez ’98 EXPSPACE exponential

Plandowski & Rytter ’98 new approach — using compression

Plandowski ’99 PSPACE $O(n^5)$

J. ’13 PSPACE $O(n \log n)$

- NP-hard, believed to be in NP
- Exact space complexity?

This talk

Word Equations are in NLinSPACE
Main idea

- Recompression algorithm [J. 2013]
- Huffman coding of letters
Main idea

- Recompression algorithm [J. 2013]
- Huffman coding of letters

The proof is more complex
- how letters depend on fragments of original equation
- special coding (so worse than Huffman)
- technically involved
Compression operations

Given a word $w$:

**$(\Sigma_\ell, \Sigma_r)$ pair compression** replace each $ab \in \Sigma_\ell \Sigma_r$ in $w$ with fresh $c_{ab}$

($\Sigma_\ell, \Sigma_r$ are disjoint)
Compression operations

Given a word $w$:

- **$(\Sigma_\ell, \Sigma_r)$ pair compression** replace each $ab \in \Sigma_\ell \Sigma_r$ in $w$ with fresh $c_{ab}$ ($\Sigma_\ell, \Sigma_r$ are disjoint)

- **$\Sigma$ block compression** replace each maximal block $a^\ell \in \Sigma^*$ in $w$ by a fresh $a^\ell$. ($a^\ell$ is a maximal block when it is in $w$ and cannot be extended by $a$).

---

We want to perform it on $S(U)$ and $S(V)$.

Occurrence can be partially in the equation and in the variable.
Compression operations

Given a word $w$:

$\left(\Sigma_{\ell}, \Sigma_{r}\right)$ pair compression replace each $ab \in \Sigma_{\ell}\Sigma_{r}$ in $w$ with fresh $c_{ab}$ ($\Sigma_{\ell}, \Sigma_{r}$ are disjoint)

$\Sigma$ block compression replace each maximal block $a_{\ell} \in \Sigma^*$ in $w$ by a fresh $a_{\ell}$. ($a_{\ell}$ is a maximal block when it is in $w$ and cannot be extended by $a$).

$\{b, c\}$ block compression
Compression operations

Given a word $w$:

- $(\Sigma_\ell, \Sigma_r)$ pair compression replace each $ab \in \Sigma_\ell \Sigma_r$ in $w$ with fresh $c_{ab}$ ($\Sigma_\ell, \Sigma_r$ are disjoint)

- $\Sigma$ block compression replace each maximal block $a^\ell \in \Sigma^*$ in $w$ by a fresh $a_\ell$. ($a^\ell$ is a maximal block when it is in $w$ and cannot be extended by $a$).

$\{b, c\}$ block compression

$aaaabbccccbbccccbbb$
Compression operations

Given a word $w$:

$(\Sigma_\ell, \Sigma_r)$ pair compression replace each $ab \in \Sigma_\ell \Sigma_r$ in $w$ with fresh $c_{ab}$
$(\Sigma_\ell, \Sigma_r$ are disjoint)

$\Sigma$ block compression replace each maximal block $a^\ell \in \Sigma^*$ in $w$ by a fresh $a_\ell$.
($a^\ell$ is a maximal block when it is in $w$ and cannot be extended by $a$).

$\{b, c\}$ block compression

\[
\text{aaaabbccbbcccbbb}
\]
\[
\text{aaab}_2b_2c_3b_2c_3b_3
\]
Compression operations

Given a word \( w \):

\((\Sigma_\ell, \Sigma_r)\) **pair compression** replace each \( ab \in \Sigma_\ell \Sigma_r \) in \( w \) with fresh \( c_{ab} \)

\((\Sigma_\ell, \Sigma_r \) are disjoint) 

\(\Sigma\) **block compression** replace each maximal block \( a^\ell \in \Sigma^* \) in \( w \) by a fresh \( a^\ell \). \((a^\ell \) is a maximal block when it is in \( w \) and cannot be extended by \( a \)).

\{b, c\} **block compression** \quad \{a, c\}, \{b\} **pair compression**

\textcolor{red}{aaabbbccccbbbbb} 
\textcolor{red}{aaab}_2 \textcolor{blue}{c_3 \ b_2 \ c_3} \textcolor{blue}{b_3}
Compression operations

Given a word $w$:

- **$(\Sigma_\ell, \Sigma_r)$ pair compression** replace each $ab \in \Sigma_\ell \Sigma_r$ in $w$ with fresh $c_{ab}$ ($\Sigma_\ell, \Sigma_r$ are disjoint)

- **$\Sigma$ block compression** replace each maximal block $a^\ell \in \Sigma^*$ in $w$ by a fresh $a_\ell$. ($a^\ell$ is a maximal block when it is in $w$ and cannot be extended by $a$).

- **$\{b, c\}$ block compression**

- **$\{a, c\}, \{b\}$ pair compression**

```latex
\begin{align*}
\text{aaabbbccccbbcccbbbb} & \\
\text{aaab}_2 \text{ c}_3 \text{ b}_2 \text{ c}_3 \text{ b}_3 & \quad \text{aaabbbccccbbcccbbbb}
\end{align*}
```
Compression operations

Given a word $w$:

- **$(\Sigma_\ell, \Sigma_r)$ pair compression** replace each $ab \in \Sigma_\ell \Sigma_r$ in $w$ with fresh $c_{ab}$ ($\Sigma_\ell, \Sigma_r$ are disjoint).

- **$\Sigma$ block compression** replace each maximal block $a^\ell \in \Sigma^*$ in $w$ by a fresh $a_\ell$. ($a^\ell$ is a maximal block when it is in $w$ and cannot be extended by $a$).

\[
\begin{align*}
\{b, c\} & \text{ block compression} & \{a, c\}, \{b\} & \text{ pair compression} \\
\text{aaabbbccbbccccbbb} & & \text{aaabbbcccbbccbb} \\
\text{aaab}_2c_3b_2c_3b_3 & & \text{aaad bcc e bcc e bb}
\end{align*}
\]
Compression operations

Given a word \( w \):

\((\Sigma_\ell, \Sigma_r)\) pair compression replace each \( ab \in \Sigma_\ell \Sigma_r \) in \( w \) with fresh \( c_{ab} \)

\((\Sigma_\ell, \Sigma_r \) are disjoint)

\(\Sigma\) block compression replace each maximal block \( a^\ell \in \Sigma^* \) in \( w \) by a fresh \( a_\ell \).

\( a^\ell \) is a maximal block when it is in \( w \) and cannot be extended by \( a \).

\{b, c\} block compression \hspace{1cm} \{a, c\}, \{b\} pair compression

\begin{align*}
\text{aaabbbcccbbeccccbb} & \quad \text{aaabbbcccbbeccccbb} \\
\text{aaab_{2}c_3b_2c_3b_3} & \quad \text{aaabbbcccbbeccccbb}
\end{align*}

\begin{itemize}
\item We want to perform it on \( S(U) \) and \( S(V) \).
\item Occurrence can be partially in the equation and in the variable.
\end{itemize}
Checking equality of two explicit words

Require: two words $u, v$ to be tested for equality

1: while $|u| > 1$ or $|v| > 1$ do
2: $\Sigma \leftarrow$ letters in $u, v$
3: perform $\Sigma$-block compression
4: while some pair in $\Sigma^2$ was not considered do
5: guess partition of $\Sigma$ to $(\Sigma_\ell, \Sigma_r)$
6: perform $(\Sigma_\ell, \Sigma_r)$ pair compression
7: test equality
Checking equality of two explicit words

Require: two words $u, v$ to be tested for equality

1: while $|u| > 1$ or $|v| > 1$ do
2: $\Sigma \leftarrow$ letters in $u, v$
3: perform $\Sigma$-block compression
4: while some pair in $\Sigma^2$ was not considered do
5: guess partition of $\Sigma$ to $(\Sigma_\ell, \Sigma_r)$
6: perform $(\Sigma_\ell, \Sigma_r)$ pair compression
7: test equality

Phase: one iteration of the main loop.
Checking equality of two explicit words

Require: two words \( u, v \) to be tested for equality

1: \( \textbf{while} \ |u| > 1 \text{ or } |v| > 1 \text{ do} \)
2: \( \Sigma \leftarrow \text{letters in } u, v \)
3: \( \text{perform } \Sigma\text{-block compression} \)
4: \( \textbf{while} \text{ some pair in } \Sigma^2 \text{ was not considered do} \)
5: \( \text{guess partition of } \Sigma \text{ to } (\Sigma_\ell, \Sigma_r) \)
6: \( \text{perform } (\Sigma_\ell, \Sigma_r) \text{ pair compression} \)
7: \( \text{test equality} \)

Phase: one iteration of the main loop.

Shortening

Consider consecutive \( ab \) in \( u, v \) at the beginning of the phase

\( a = b \) compressed as a block

\( a \neq b \) considered and compressed, or
one of them was compressed earlier
In a solution word $S(U)$ or $S(V)$:

- pair is from the equation: OK, we replace it
Pair Compression on word equation

In a solution word $S(U)$ or $S(V)$:

- pair is from the equation: OK, we replace it
- it is from the substitution for a variable: OK, solution changes
Pair Compression on word equation

In a solution word $S(U)$ or $S(V)$:

- pair is from the equation: OK, we replace it
- it is from the substitution for a variable: OK, solution changes
- partially here and there: just pop the problematic letter out
Pair Compression on word equation

In a solution word $S(U)$ or $S(V)$:

- pair is from the equation: OK, we replace it
- it is from the substitution for a variable: OK, solution changes
- partially here and there: just pop the problematic letter out

**PairCompression(Σ$\ell$, Σ$r$)**

1: for $X \in X$ do
2: let $b$: first letter of $S(X)$ ▶ Guess
3: if $b \in Σ_r$ then
4: replace each occurrence of $X$ by $bX$ ▶ Pop
5: if $S(X) = \epsilon$ then ▶ Guess
6: remove $X$ from the equation
7: let $a$: last . . . ▶ symmetrically for the last letter and $Σ_ℓ$
8: perform pair compression on sides of the equation
BlockCompression

1: for $X \in \mathcal{X}$ do
2: let $S(X) = a^\ell w b^r$
3: replace $X$ with $a^\ell X b^r$
4: if $S(X) = \epsilon$ then ▶ Guess
5: remove $X$ from the equation
6: perform block compression on sides of the equation
The algorithm

### Main algorithm

1. **while** sides of the equation are nontrivial **do**
2. \( \Sigma \leftarrow \text{letters in the equation} \)
3. perform \( \Sigma \)-block compression
4. **while** some pair in \( \Sigma^2 \) was not considered **do**
5. **guess** partition of \( \Sigma \) to \((\Sigma_\ell, \Sigma_r)\)  

\[ \triangleright \text{Important} \]
6. perform \((\Sigma_\ell, \Sigma_r)\) pair compression
### Main algorithm

1. **while** sides of the equation are nontrivial **do**
2. \( \Sigma \leftarrow \) letters in the equation
3. perform \( \Sigma \)-block compression
4. **while** some pair in \( \Sigma^2 \) was not considered **do**
5. guess partition of \( \Sigma \) to \( (\Sigma_\ell, \Sigma_r) \) ▶ Important
6. perform \( (\Sigma_\ell, \Sigma_r) \) pair compression

A **phase** is one iteration of the main loop
### Main algorithm

1: while sides of the equation are nontrivial do
2: \[ \Sigma \leftarrow \text{letters in the equation} \]
3: perform \( \Sigma \)-block compression
4: while some pair in \( \Sigma^2 \) was not considered do
5: guess partition of \( \Sigma \) to \((\Sigma_\ell, \Sigma_r)\)
6: perform \((\Sigma_\ell, \Sigma_r)\) pair compression

---

A **phase** is one iteration of the main loop

### Encoding

We use Huffman coding for letters. (Need to recalculate it.)

We use different encoding in the analysis.
We modify the equation, but think that we operate on \( S(U) = S(V) \). We fix a solution for a phase.
We modify the equation, but think that we operate on \( S(U) = S(V) \). We fix a solution for a phase.

In NLinSPACE we can analyse only “good choices”: if we exceed the space then we reject.
Definition (Dependency interval)

An interval of positions in the input equation is called a dependency interval (depint).
We associate a depint to each symbol in the equation; $D = \text{dep}(p)$. 

$p$

$D = \text{dep}(p)$
Definition (Dependency interval)

An interval of positions in the input equation is called a dependency interval (depint). We associate a depint to each symbol in the equation; \( D = \text{dep}(p) \).

Assigning depints

- Technical, operational manner.
- We expand the depints by taking unions with neighbouring ones.
- Popped letters have depints of their variables.
- Depints of letters introduced due to compression do not change.
Depints and encoding

Encoding

- letter at position $p \rightarrow UV[\text{dep}(p)]$
- letters with this interval assigned are numbered $1, 2, \ldots, k$
- we assign to them codes $UV[D]#1, UV[D]#2, \ldots, UV[D]#k$
Encoding

- letter at position $p \rightarrow UV[\text{dep}(p)]$
- letters with this interval assigned are numbered $1, 2, \ldots, k$
- we assign to them codes $UV[D]\#1, UV[D]\#2, \ldots, UV[D]\#k$
Depints and encoding

**Encoding**

- letter at position \( p \) → \( UV[\text{dep}(p)] \)
- letters with this interval assigned are numbered 1, 2, \ldots, \( k \)
- we assign to them codes \( UV[D]\#1, UV[D]\#2, \ldots, UV[D]\#k \)
Depints and encoding

Encoding

- letter at position $p \rightarrow UV[\text{dep}(p)]$
- letters with this interval assigned are numbered $1, 2, \ldots, k$
- we assign to them codes $UV[D]\#1, UV[D]\#2, \ldots, UV[D]\#k$
- formally not encoding: assigns different codes to the same letter
- never assigns the same code to different letters
- worse than Huffman coding; enough to estimate its bit-size
Depints: positions to indices
Depints size

Depints: positions to indices
Depints: positions to indices
Depints: positions to indices
Index to positions \text{Pos}(i)
Depints size

Depints: positions to indices
Index to positions $\text{Pos}(i)$
$\text{Pos}(i)$ are intervals
Depints: positions to indices
Index to positions $\text{Pos}(i)$
$\text{Pos}(i)$ are intervals
$\text{Pos}(i)$ grows: extending, popping letters
Depints: positions to indices
Index to positions $\text{Pos}(i)$
$\text{Pos}(i)$ are intervals
$\text{Pos}(i)$ grows: extending, **popping letters**
Depints: positions to indices
Index to positions $\text{Pos}(i)$
$\text{Pos}(i)$ are intervals
$\text{Pos}(i)$ grows: extending, popping letters
$\text{Pos}(i)$ shrinks: compressions
Depints: positions to indices
Index to positions Pos(i)
Pos(i) are intervals
Pos(i) grows: extending, popping letters
Pos(i) shrinks: compressions
Depints: positions to indices
Index to positions $\text{Pos}(i)$
$\text{Pos}(i)$ are intervals
$\text{Pos}(i)$ grows: extending, popping letters
$\text{Pos}(i)$ shrinks: compressions
Fresh letters block:
Letter to the left of $\text{Pos}(i)$ is new — no extensions
Depints: positions to indices
Index to positions $\text{Pos}(i)$
$\text{Pos}(i)$ are intervals
$\text{Pos}(i)$ grows: extending, popping letters
$\text{Pos}(i)$ shrinks: compressions
Fresh letters block:
Letter to the left of $\text{Pos}(i)$ is new — no extensions
Depints: positions to indices
Index to positions $\text{Pos}(i)$
$\text{Pos}(i)$ are intervals
$\text{Pos}(i)$ grows: extending, popping letters
$\text{Pos}(i)$ shrinks: compressions
Fresh letters block:
Letter to the left of $\text{Pos}(i)$ is new — no extensions
Left letter in $S(X)$ is new — no popping
How to choose partitions

- Our only choice that affects size is the partition.
- Choose the partitions to minimise bit size.
- If \( \text{Pos}(i) = \mathcal{O}(1) \) then everything works.
How to choose partitions

- Our only choice that affects size is the partition.
- Choose the partitions to minimise bit size.
- If $\text{Pos}(i) = \mathcal{O}(1)$ then everything works.

Random partition to expectation

- Random compresses a pair with probability $1/4$.
- Each blocking is with probability $1/4$.
- Turn this into expectation: calculate what to minimise: length, frequency, new letters, number of occurrences, ...

\[
\sum_{i \geq 0} \frac{1}{2^i} = 2
\]

\[
\sum_{i \geq 0} \frac{i^2 \log i}{2^i} = \mathcal{O}(1)
\]
Other technicalities

Some other technicalities

- need to change Huffman coding
- how to make block compression (no explicit numbers — known)
- what happens with the solution
- ending markers with special treatment
- ...

Works for Human coding of the input.
Other technicalities

Some other technicalities

- need to change Huffman coding
- how to make block compression (no explicit numbers — known)
- what happens with the solution
- ending markers with special treatment
- ...

Works for Huffman coding of the input.
Open questions

- Are word equations in NP?
- Can this be generalised to other equations? (constraints, involution, commutation)