Uniwersytet Wrocławski

Word equations in nondeterministic linear space

Artur Jeż
Institute of Computer Science

International Colloquium on Automata, Languages and Programming
Warszawa, 13.07.2017

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?

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?

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

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?

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

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?

$$a\,X\,b\,X\,Y\,bbb = X\,abaab\,Y\,b\,Y \qquad S(X) = aa, S(Y) = bb$$
$$aaabaabbbbb = aaabaabbbbb$$

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

$$a\,X\,b\,X\,Y\,bbb = X\,abaab Y\,b Y \qquad S(X) = aa, S(Y) = bb$$
$$aaabaabbbbb = aaabaabbbbb$$

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

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 $\mathcal{O}(n^5)$

J. '13 PSPACE $\mathcal{O}(n \log n)$

- 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 $\mathcal{O}(n^5)$

J. '13 PSPACE $\mathcal{O}(n \log n)$

- 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:

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

Compression operations

Given a word w:

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

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

Compression operations

Given a word w:

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

Σ **block compression** replace each maximal block $a^\ell \in \Sigma^*$ in w by a fresh a_ℓ. (a^ℓ 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:

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

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

$\{b, c\}$ block compression

$$aaabbcccbbcccbbb$$

Compression operations

Given a word w:

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

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

$\{b, c\}$ block compression

$$aaabbcccbbcccbbb$$
$$aaab_2\ c_3\ b_2\ c_3\ \ b_3$$

Compression operations

Compression operations

Given a word w:

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

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

$\{b, c\}$ block compression

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

$$aaabbcccbbcccbbb$$
$$aaab_2\, c_3\, b_2\, c_3\, b_3$$

Compression operations

Compression operations

Given a word w:

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

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

$\{b, c\}$ block compression

$$aaabbcccbbcccbbb$$
$$aaab_2\ c_3\ b_2\ c_3\ \ b_3$$

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

$$aaabbcccbbcccbbb$$

Compression operations

Given a word w:

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

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

$\{b, c\}$ block compression

$$aaabbcccbbcccbbb$$
$$aaab_2\ c_3\ b_2\ c_3\ \ b_3$$

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

$$aaabbcccbbcccbbb$$
$$aa\ d\ bcc\ e\ bcc\ e\ bb$$

Compression operations

Given a word w:

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

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

$\{b, c\}$ block compression

$$aaabbcccbbcccbbb$$
$$aaab_2\ c_3\ b_2\ c_3\ \ b_3$$

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

$$aaabbcccbbcccbbb$$
$$aa\ d\ bcc\ e\ bcc\ e\ bb$$

- We want to perform it on $S(U)$ and $S(V)$.
- Occurrence can be partially in the equation and in the variable.

Preliminaries: explicit word

Checking equality of two explicit words

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

```
while |u| > 1 or |v| > 1 do
    Σ ← letters in u, v
    perform Σ-block compression
    while some pair in Σ² was not considered do
        guess partition of Σ to (Σ_ℓ, Σ_r)
        perform (Σ_ℓ, Σ_r) pair compression
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 Σ-block compression
4: **while** some pair in Σ^2 was not considered **do**
5: guess partition of Σ to (Σ_ℓ, Σ_r)
6: perform (Σ_ℓ, Σ_r) pair compression
7: test equality

Phase: one iteration of the main loop.

Preliminaries: explicit word

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: $\quad\Sigma \leftarrow$ letters in u, v
3: $\quad$perform Σ-block compression
4: $\quad$**while** some pair in Σ^2 was not considered **do**
5: $\quad\quad$guess partition of Σ to (Σ_ℓ, Σ_r)
6: $\quad\quad$perform (Σ_ℓ, Σ_r) pair compression
7: 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

Pair Compression on word equation

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_ℓ, Σ_r)

1: **for** $X \in \mathcal{X}$ **do**
2: let b: first letter of $S(X)$ ▷ Guess
3: **if** $b \in \Sigma_r$ **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

Block Compression

BlockCompression

for $X \in \mathcal{X}$ **do**

  let $S(X) = a^\ell w b^r$ ▷ Guess

  replace X with $a^\ell X b^r$

  if $S(X) = \epsilon$ **then** ▷ Guess

    remove X from the equation

perform block compression on sides of the equation

The algorithm

Main algorithm

1: **while** sides of the equation are nontrivial **do**
2: $\Sigma \leftarrow$ letters in the equation
3: perform Σ-block compression
4: **while** some pair in Σ^2 was not considered **do**
5: guess partition of Σ to (Σ_ℓ, Σ_r) ▷ Important
6: perform (Σ_ℓ, Σ_r) pair compression

The algorithm

Main algorithm

while sides of the equation are nontrivial **do**
    $\Sigma \leftarrow$ letters in the equation
    perform Σ-block compression
    while some pair in Σ^2 was not considered **do**
        guess partition of Σ to (Σ_ℓ, Σ_r) ▷ Important
        perform (Σ_ℓ, Σ_r) pair compression

A phase is one iteration of the main loop

The algorithm

Main algorithm

1: **while** sides of the equation are nontrivial **do**
2: $\Sigma \leftarrow$ letters in the equation
3: perform Σ-block compression
4: **while** some pair in Σ^2 was not considered **do**
5: guess partition of Σ to (Σ_ℓ, Σ_r) ▷ Important
6: perform (Σ_ℓ, Σ_r) pair compression

A phase is one iteration of the main loop

Encoding

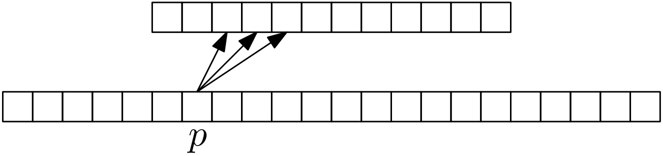
We use Huffman coding for letters. (Need to recalculate it.)
We use different encoding in the analysis.

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.

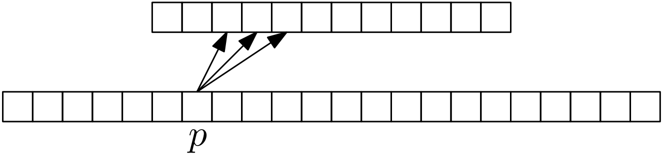
Dependency interval



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; $D = \mathrm{dep}(p)$.

Dependency interval



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; $D = \mathrm{dep}(p)$.

Assigning depints

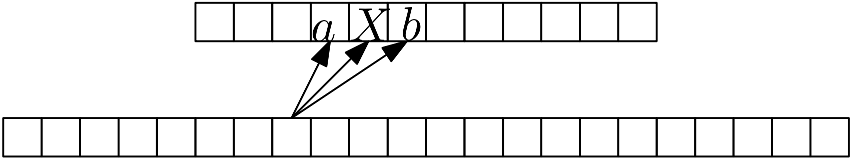
- Technical, operational manner.
- We expand the depints by taking unions with neighbouring ones.
- Popped letters have depints of their variables.
- Depints of letters introduced due to compression do not change.

Depints and encoding

Encoding

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

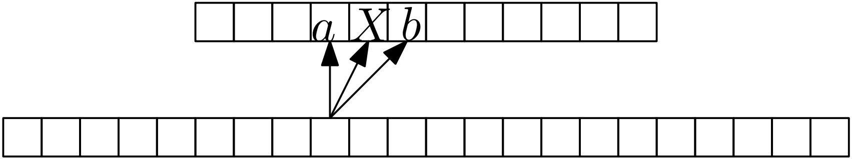
Depints and encoding



Encoding

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

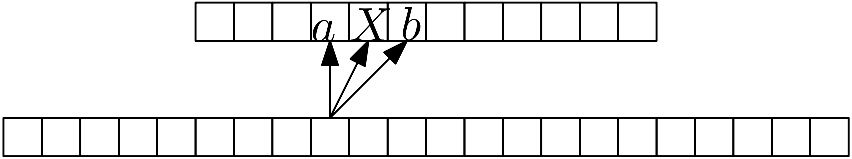
Depints and encoding



Encoding

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

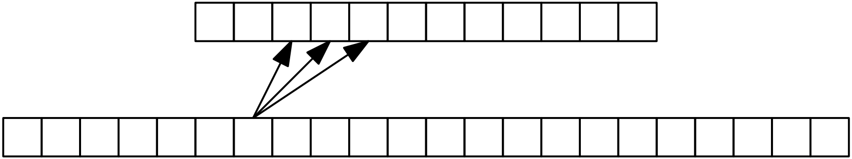
Depints and encoding



Encoding

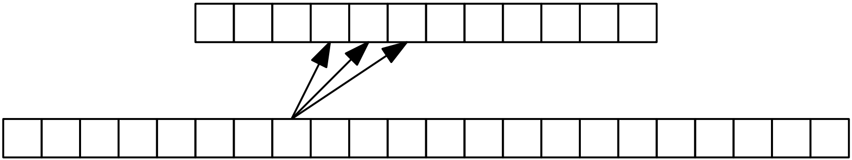
- letter at position $p \to UV[\mathrm{dep}(p)]$
- letters with this interval assigned are numbered $1, 2, \ldots, k$
- we assign to them codes $UV[D]\#1, UV[D]\#2, \ldots, UV[D]\#k$
- formally not encoding: assigns different codes to the same letter
- never assigns the same code to different letters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

Depints size



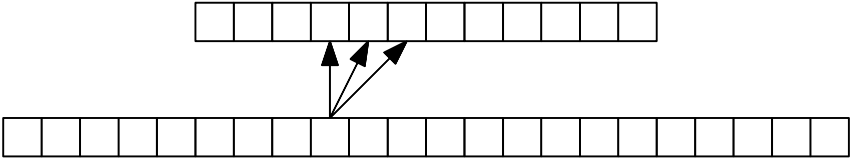
Depints: positions to indices

Depints size



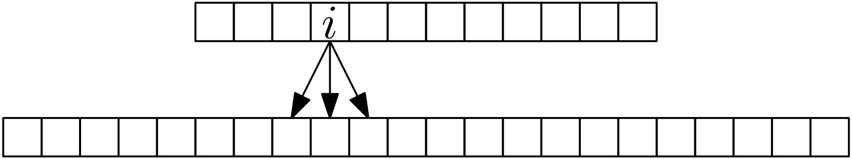
Depints: positions to indices

Depints size



Depints: positions to indices

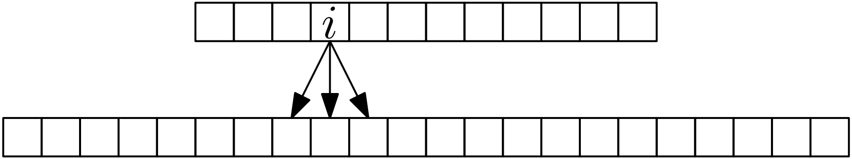
Depints size



Depints: positions to indices

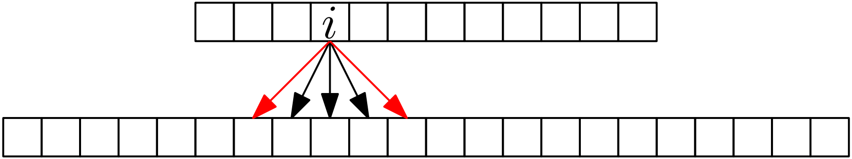
Index to positions Pos(i)

Depints size



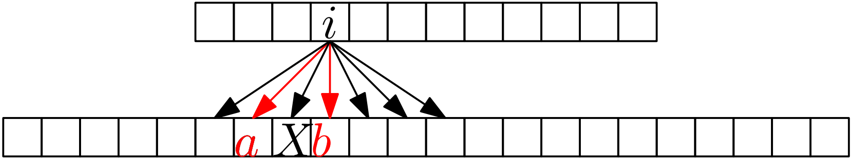
Depints: positions to indices
Index to positions Pos(i)
Pos(i) are intervals

Depints size



Depints: positions to indices
Index to positions Pos(i)
Pos(i) are intervals
Pos(i) grows: extending, popping letters

Depints size



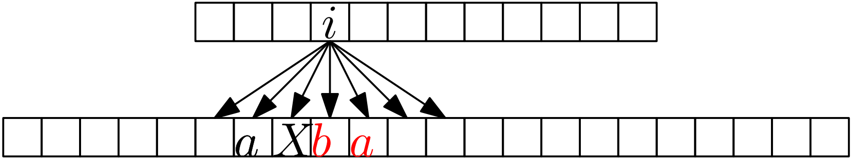
Depints: positions to indices

Index to positions Pos(i)

Pos(i) are intervals

Pos(i) grows: extending, popping letters

Depints size



Depints: positions to indices

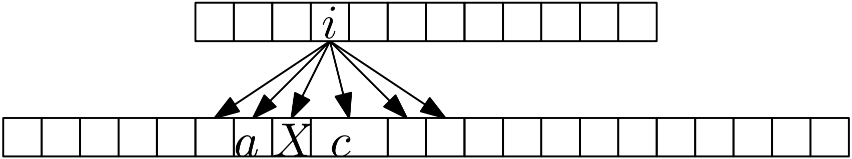
Index to positions Pos(i)

Pos(i) are intervals

Pos(i) grows: extending, popping letters

Pos(i) shrinks: compressions

Depints size



Depints: positions to indices

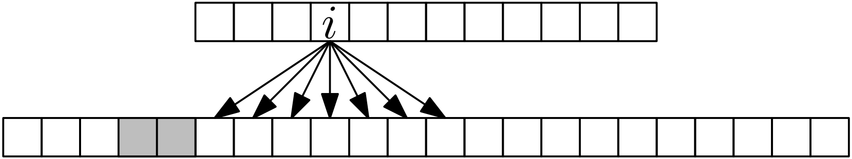
Index to positions Pos(i)

Pos(i) are intervals

Pos(i) grows: extending, popping letters

Pos(i) shrinks: compressions

Depints size



Depints: positions to indices

Index to positions Pos(i)

Pos(i) are intervals

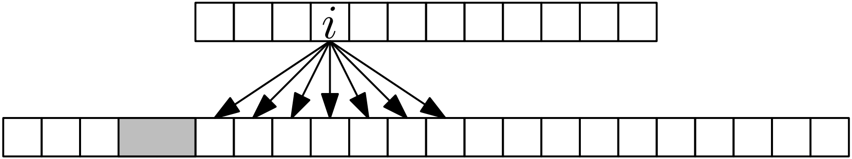
Pos(i) grows: extending, popping letters

Pos(i) shrinks: compressions

Fresh letters block:

Letter to the left of Pos(i) is new — no extensions

Depints size



Depints: positions to indices

Index to positions Pos(i)

Pos(i) are intervals

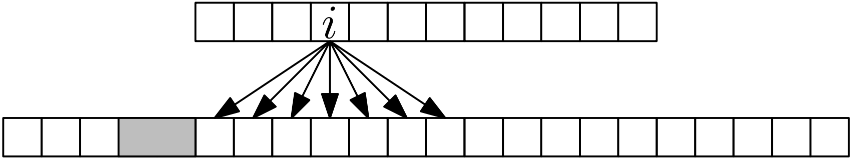
Pos(i) grows: extending, popping letters

Pos(i) shrinks: compressions

Fresh letters block:

Letter to the left of Pos(i) is new — no extensions

Depints size



Depints: positions to indices

Index to positions Pos(i)

Pos(i) are intervals

Pos(i) grows: extending, popping letters

Pos(i) shrinks: compressions

Fresh letters block:

Letter to the left of 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 $\mathsf{Pos}(i) = \mathcal{O}(1)$ then everything works.

How to choose partitions

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

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)