One-variable word equations in linear time

Artur Jeż

MPI Saarbrücken

08.07.2013

Word Equations

Definition

Given equation $\mathcal{A} = \mathcal{B}$, where $\mathcal{A}, \mathcal{B} \in (\Sigma \cup \mathcal{X})^*$.
Is there an assignment $S : \mathcal{X} \mapsto \Sigma^*$ satisfying the equation?

- in PSPACE
- NP-hard

Word Equations

Definition

Given equation $\mathcal{A} = \mathcal{B}$, where $\mathcal{A}, \mathcal{B} \in (\Sigma \cup \mathcal{X})^*$.
Is there an assignment $S : \mathcal{X} \mapsto \Sigma^*$ satisfying the equation?

- in PSPACE
- NP-hard

One variable

Return all solutions.

- naively: $\mathcal{O}(n^3)$
- $\mathcal{O}(n \log n)$ [Obono, Goralcik and Maksimenko '94]
- $\mathcal{O}(n + \#_X \log n)$ [Dąbrowski and Plandowski '99]

Results

New algorithm for one variable

- based on recompression [applicable to general case]
- running time $\mathcal{O}(n + \#_X \log n)$

Results

New algorithm for one variable

- based on recompression [applicable to general case]
- running time $\mathcal{O}(n + \#_X \log n)$
- $\mathcal{O}(n)$
  - heuristics
  - data structures (suffix-arrays, longest common prefix queries)
  - word combinatorics
  - better analysis

Univariate equations

Form of the equation $\mathcal{A} = \mathcal{B}$

$$A_0XA_1 \ldots A_{k-1}XA_k = XB_1 \ldots B_{\ell-1}XB_\ell,$$

$$\text{where } A_i, B_i \in \Sigma^*,\ A_0 \neq \epsilon.$$

Univariate equations

Form of the equation $\mathcal{A} = \mathcal{B}$

$$A_0 X A_1 \dots A_{k-1} X A_k = X B_1 \dots B_{\ell-1} X B_\ell,$$
$$\text{where } A_i, B_i \in \Sigma^*,\ A_0 \neq \epsilon.$$

Want only $S(X) \neq \epsilon$

Write $S(\mathcal{A})$ and $S(\mathcal{B})$.

Univariate equations

Form of the equation $\mathcal{A} = \mathcal{B}$

$$A_0 X A_1 \dots A_{k-1} X A_k = X B_1 \dots B_{\ell-1} X B_\ell,$$
$$\text{where } A_i, B_i \in \Sigma^*,\ A_0 \neq \epsilon.$$

Want only $S(X) \neq \epsilon$

Write $S(\mathcal{A})$ and $S(\mathcal{B})$.

Properties

- first (last) letter of $S(X)$ is known
- $S(X) = A_0^i A'$, where A' is a prefix of A_0 (trivial)
  if $A_0 \in a^+$ then $S(X) \in a^+$
- testing solutions in a^* is simple (linear time)

Equality and Compression of Strings

$$a\ a\ a\ b\ a\ b\ c\ a\ b\ a\ b\ b\ a\ b\ c\ b\ a$$

$$a\ a\ a\ b\ a\ b\ c\ a\ b\ a\ b\ b\ a\ b\ c\ b\ a$$

Equality and Compression of Strings

$$\color{red}{a\ a\ a}\ b\ a\ b\ c\ a\ b\ a\ b\ b\ a\ b\ c\ b\ a$$

$$\color{red}{a\ a\ a}\ b\ a\ b\ c\ a\ b\ a\ b\ b\ a\ b\ c\ b\ a$$

Equality and Compression of Strings

$a_3 \; b \; a \; b \; c \; a \; b \; a \; b \; b \; a \; b \; c \; b \; a$

$a_3 \; b \; a \; b \; c \; a \; b \; a \; b \; b \; a \; b \; c \; b \; a$

Equality and Compression of Strings

$$a_3 \quad b\ a\ b\ c\ a\ b\ a \quad b_2\ a\ b\ c\ b\ a$$

$$a_3 \quad b\ a\ b\ c\ a\ b\ a \quad b_2\ a\ b\ c\ b\ a$$

Equality and Compression of Strings

$$a_3 \quad b \quad \color{green}{d} \quad c \quad \color{green}{d} \quad a \quad \color{blue}{b_2} \quad \color{green}{d} \quad c \; \color{red}{b} \; \color{red}{a}$$

$$a_3 \quad b \quad \color{green}{d} \quad c \quad \color{green}{d} \quad a \quad \color{blue}{b_2} \quad \color{green}{d} \quad c \; \color{red}{b} \; \color{red}{a}$$

Equality and Compression of Strings

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

Equality and Compression of Strings

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

$$a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e$$

Iterate!

Compression

1: $P \leftarrow$ all pairs from $S(\mathcal{A})$, $L \leftarrow$ all letters from $S(\mathcal{A})$
2: **for** each $a \in L$ **do**
3: replace each maximal block a^ℓ by a_ℓ ▷ A fresh letter
4: **for** each $ab \in P$ **do**
5: replace each ab by c ▷ A fresh letter

Compression

1: $P \leftarrow$ all pairs from $S(\mathcal{A})$, $L \leftarrow$ all letters from $S(\mathcal{A})$
2: **for** each $a \in L$ **do**
3: replace each maximal block a^ℓ by a_ℓ ▷ A fresh letter
4: **for** each $ab \in P$ **do**
5: replace each ab by c ▷ A fresh letter

Lemma

Each subword shortens by a constant factor (A_i, B_j, $S(X)$, $S(\mathcal{A})$, $\ldots$).

Proof.

Two consecutive letters: we tried to compress them;
fail: one is already compressed. □

Compression of pairs

Type of pair

Pair appearances in $S(\mathcal{A})$:

- explicit letters,
- implicit (from $S(X)$),
- crossing: one letter explicit, one from $S(X)$

ab is crossing if it has a crossing appearance, non-crossing otherwise.

Compression of pairs

Type of pair

Pair appearances in $S(\mathcal{A})$:

- explicit letters,
- implicit (from $S(X)$),
- crossing: one letter explicit, one from $S(X)$

ab is crossing if it has a crossing appearance, non-crossing otherwise.

Consider $aabXacXdeX = XaabacXdeX$ under $S(X) = aab$

- $aabaabacaabdeaab$ $[aabXacXdeX]$
- $aabaabacaabdeaab$ $[aabXacXdeX]$
- $aabaabacaabdeaab$ $[aabXacXdeX]$

Crossing pairs: ba, ca, bd, ea.

Non-crossing pair compression

Replace each explicit ab by a fresh letter (in $S(X)$: implicitly).

- $aabXacXdeX = XaabacXdeX$ with $S(X) = aab$
- replace ab by f
- $afXacXdeX = XafacXdeX$ with $S(X) = af$

Non-crossing pair compression

Replace each explicit ab by a fresh letter (in $S(X)$: implicitly).

- $aabXacXdeX = XaabacXdeX$ with $S(X) = aab$
- replace ab by f
- $afXacXdeX = XafacXdeX$ with $S(X) = af$

Crossing pair

When ab is 'crossing' because of aX then replace X with bX (similar for Xb and XX).

Lemma

After this the pair stops to be crossing.

Example

- $ababab X b X = X babab X ba$
- $S(X) = (ab)^i$ or $S(X) = (ab)^i a$
- the former is not possible ($S(X)$ ends with a)
- ab is crossing: replace each X with Xa [test $S(X) = a$]
- $ababab X ab X a = X ababab X aba$ with $S(X) = (ab)^i$
- ab is non-crossing: replace each ab with c
- $ccc X c X = X ccc X c$ with $S(X) = c^i$ (trivial case!)

Blocks

The same for blocks:

- replace maximal blocks
- explicit, implicit, crossing appearances
- crossing blocks, noncrossing blocks
- cutting a-prefixes and a-suffixes
- then a is without crossing blocks

Algorithm

while $A_0 \notin a^*$ **do**
    $L \leftarrow$ all letters from $S(\mathcal{A})$
    for $a \in L$ **do**
        uncross and compress a blocks
    $P \leftarrow$ non-crossing pairs from $S(\mathcal{A})$, $P' \leftarrow$ crossing
    for each $ab \in P$ **do**
        compress ab
    for each $ab \in P'$ **do**
        uncross and compress ab

Algorithm

while $A_0 \notin a^*$ **do**
  $L \leftarrow$ all letters from $S(\mathcal{A})$
  for $a \in L$ **do**
    uncross and compress a blocks
  $P \leftarrow$ non-crossing pairs from $S(\mathcal{A})$, $P' \leftarrow$ crossing
  for each $ab \in P$ **do**
    compress ab
  for each $ab \in P'$ **do**
    uncross and compress ab

Whenever we uncross, we test a solution.

Shortening

Definition

An A_i (B_j) is short if it has length at most 100 and is long otherwise.

Shortening

Definition

An A_i (B_j) is short if it has length at most 100 and is long otherwise.

Lemma

If A_i is long then its length decreases by $1/4$ in a phase.
If it is short than it stays short.

Proof.

- $\mathcal{O}(1)$ letters are introduced due to uncrossing.
- A_i is compressed by a constant
- $len_{k+1} = \frac{3}{4} len_k + c$ □

Shortening

Definition

An A_i (B_j) is short if it has length at most 100 and is long otherwise.

Lemma

If A_i is long then its length decreases by $1/4$ in a phase.
If it is short than it stays short.

Proof.

- $\mathcal{O}(1)$ letters are introduced due to uncrossing.
- A_i is compressed by a constant
- $len_{k+1} = \frac{3}{4} len_k + c$ □

A_0 always decreases by $1/4$ in a phase.

Simple charging

- One phase takes linear time
  - compression: grouping by RadixSort
  - verification: naive, $\mathcal{O}(1)$ candidates

Simple charging

- One phase takes linear time
  - compression: grouping by RadixSort
  - verification: naive, $\mathcal{O}(1)$ candidates
- Charge towards the words.

  long looses constant fraction of length, charge it.
  $\mathcal{O}(n)$ in total

Simple charging

- One phase takes linear time
  - compression: grouping by RadixSort
  - verification: naive, $\mathcal{O}(1)$ candidates
- Charge towards the words.

  long: looses constant fraction of length, charge it.
  $\mathcal{O}(n)$ in total

  short: We charge only $\mathcal{O}(1)$ to it.
  $\mathcal{O}(\log |A_0|)$ phases.
  $\mathcal{O}(\#_X \log |A_0|)$ in total.

Simple charging

- One phase takes linear time
  - compression: grouping by RadixSort
  - verification: naive, $\mathcal{O}(1)$ candidates
- Charge towards the words.

  long: looses constant fraction of length, charge it.
  $\mathcal{O}(n)$ in total

  short: We charge only $\mathcal{O}(1)$ to it.
  $\mathcal{O}(\log|A_0|)$ phases.
  $\mathcal{O}(\#_X \log|A_0|)$ in total.

The only problem: short words (compression and testing).

Towards a better charging

Separately

- storage (compression)
- testing

Towards a better charging

Separately

- storage (compression)
- testing

Lemma (Easy solutions)

If solution S is of the form v^k, where $|v| \in \mathcal{O}(1)$ then the algorithm reports it in $\mathcal{O}(1)$ phases.

Towards a better charging

Separately

- storage (compression)
- testing

Lemma (Easy solutions)

If solution S is of the form v^k, where $|v| \in \mathcal{O}(1)$ then the algorithm reports it in $\mathcal{O}(1)$ phases.

Proof.

- imagine each v is compressed independently
- v reduced to a single letter
- block replaced $\square$

Storage

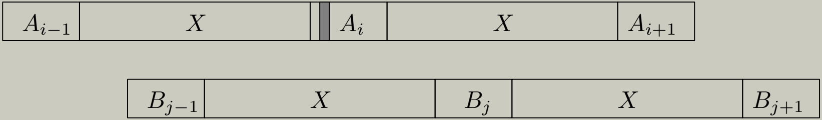
- store each short word once (pointers)
  if two short words are (non-)equal they stay (non-)equal
- substrings of long words: size proportional to long words

Storage

- store each short word once (pointers)
  if two short words are (non-)equal they stay (non-)equal
- substrings of long words: size proportional to long words
- When not? Then $S(X)$ is easy: reported in $\mathcal{O}(1)$ phases

Testing

Cost of individual test.

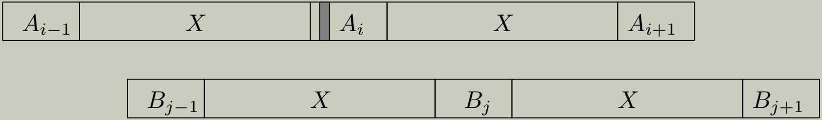


Comparison for letter in A_i

- If any of A_i, B_j or four neighbours are long: fine.
- only the case in which all are short

Testing

Cost of individual test.

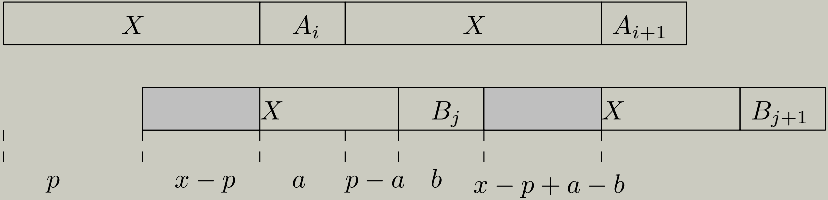


Comparison for letter in A_i

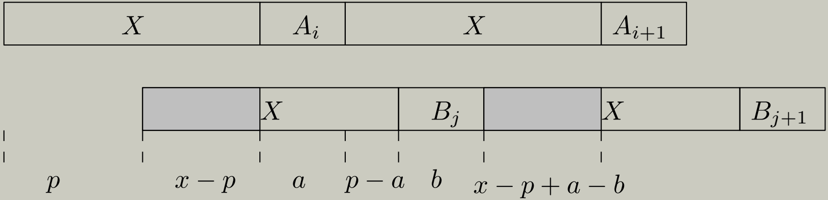
- If any of A_i, B_j or four neighbours are long: fine.
- only the case in which all are short

- Four different type of tests
- in three of them amortised cost is $\mathcal{O}(1)$ per word (in total)
- one non-trivial

Nontrivial case

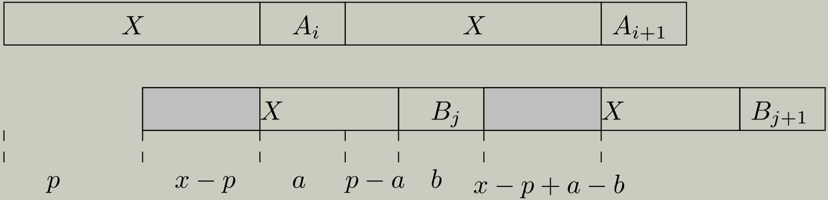


Nontrivial case



- $S(X)$ is easy
- $S(X)$ was easy when last of A_i, A_{i+1}, B_j, B_{j+1} became short

Nontrivial case



- $S(X)$ is easy
- $S(X)$ was easy when last of A_i, A_{i+1}, B_j, B_{j+1} became short
- so it was tested $\mathcal{O}(1)$ phases afterwards
- $\mathcal{O}(1)$ cost per word in total

Question and comments

Word equations

two variables All known algorithm are very complicated.
Can this approach work faster?

general In general case this is PSPACE. In NP?

Recompression technique

- general word equations
- compressed pattern matching
- approximation of the smallest grammar
- fully compressed membership problem
- ?