Fully compressed pattern matching by recompression

Artur Jeż
University of Wrocław

9 VII 2012

SLP

Definition (SLP: Straight Line Programme)

CFG generating exactly one word

$X_i \to X_j X_k$ or $X_i \to a$

SLP

Definition (SLP: Straight Line Programme)

CFG generating exactly one word
$X_i \to X_j X_k$ or $X_i \to a$

Example

$X_0 = a$, $X_1 = b$, $X_{n+1} = X_{n-1} X_{n-2}$
a, b, ba, bab, $babba$, $babbababb$, ...

SLP

Definition (SLP: Straight Line Programme)

CFG generating exactly one word
$X_i \to X_j X_k$ or $X_i \to a$

Example

$X_0 = a$, $X_1 = b$, $X_{n+1} = X_{n-1} X_{n-2}$
a, b, ba, bab, $babba$, $babbababb$, ...

Relations to LZ and LZW

LZW rules $X_i \to aX_j$, text is $X_1 X_2 X_3 \ldots$

LZ LZ to SLP: from n to $\mathcal{O}(n \log(N/n))$

SLP

Definition (SLP: Straight Line Programme)

CFG generating exactly one word
$X_i \to X_j X_k$ or $X_i \to a$

Example

$X_0 = a$, $X_1 = b$, $X_{n+1} = X_{n-1} X_{n-2}$
a, b, ba, bab, $babba$, $babbababb$, $\dots$

Relations to LZ and LZW

LZW rules $X_i \to aX_j$, text is $X_1 X_2 X_3 \dots$
LZ LZ to SLP: from n to $\mathcal{O}(n \log(N/n))$

- many algorithms for SLPs
- CPM for LZ [Gawrychowski ESA'11]
- in theory (word equations, equations in groups, verification...)

This talk

Definition (CPM, FCPM)

Compressed pattern matching: text is compressed, pattern not.
Fully Compressed pattern matching: both text and pattern are compressed.

This talk

Definition (CPM, FCPM)

Compressed pattern matching: text is compressed, pattern not.
Fully Compressed pattern matching: both text and pattern are compressed.

Results

An $\mathcal{O}((n+m)\log M)$ algorithm for FCPM for SLP.
(Previously: $\mathcal{O}(nm^2)$, [Lifshits, CPM'07]).

This talk

Definition (CPM, FCPM)

Compressed pattern matching: text is compressed, pattern not.
Fully Compressed pattern matching: both text and pattern are compressed.

Results

An $\mathcal{O}((n+m)\log M)$ algorithm for FCPM for SLP.
(Previously: $\mathcal{O}(nm^2)$, [Lifshits, CPM'07]).

Different approach

A new technique; recompression.

- decompresses text and pattern
- compresses them again (in the same way)
- in the end: pattern is a single symbol

Technique

Where it comes from

Mehlhorn, Gawry

Technique

Where it comes from

Mehlhorn, Gawry

Applicable to

- Fully Compressed Membership Problem [$\in$ NP]
- Word equations [alternative PSPACE algorithm]
- Fully Compressed Pattern Matching [SLPs, LZ, $\mathcal{O}((n+m)\log M \log(n+m))$]
- construction of a grammar for a string [alternative $\log(N/n)$ approximation algorithm]
- other?

Example

Equality of strings

How to test equality of strings?

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

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

Example

Equality of strings

How to test equality of strings?

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

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

Example

Equality of strings

How to test equality of strings?

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

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

Example

Equality of strings

How to test equality of strings?

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

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

Example

Equality of strings

How to test equality of strings?

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

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

Example

Equality of strings

How to test equality of strings?

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

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

Example

Equality of strings

How to test equality of strings?

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

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

Example

Equality of strings

How to test equality 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!

How to generalise?

Idea

For both strings

- replace pairs of letters
- replace (maximal) blocks of the same letter

When every letter is compressed, the length reduces by half in an iteration.

How to generalise?

Idea

For both strings

- replace pairs of letters
- replace (maximal) blocks of the same letter

When every letter is compressed, the length reduces by half in an iteration.

TODO

- formalise
- for SLPs
- for pattern matching
- running time

Formalisation

In one phase

Formalisation

In one phase

- $L \leftarrow$list of letters, $P \leftarrow$list of pairs of letters

Formalisation

In one phase

- $L \leftarrow$list of letters, $P \leftarrow$list of pairs of letters
- **for** every letter $a \in L$ **do**
  replace (maximal) blocks a^ℓ with a_ℓ

Formalisation

In one phase

- $L \leftarrow$list of letters, $P \leftarrow$list of pairs of letters
- **for** every letter $a \in L$ **do**
  replace (maximal) blocks a^ℓ with a_ℓ
- **for** every pair of letter $ab \in P$ **do**
  replace pairs ab with c

Formalisation

In one phase

- $L \leftarrow$list of letters, $P \leftarrow$list of pairs of letters
- **for** every letter $a \in L$ **do**
  replace (maximal) blocks a^ℓ with a_ℓ
- **for** every pair of letter $ab \in P$ **do**
  replace pairs ab with c

It will shorten the strings by constant factor.

Formalisation

In one phase

- $L \leftarrow$list of letters, $P \leftarrow$list of pairs of letters
- **for** every letter $a \in L$ **do**
  replace (maximal) blocks a^ℓ with a_ℓ
- **for** every pair of letter $ab \in P$ **do**
  replace pairs ab with c

It will shorten the strings by constant factor.

Loop, while nontrivial.
($\mathcal{O}(\log M)$ iterations).

SLPs

Grammar form

More general rules: $X_i \to uX_jvX_kw, \quad j, k < i.$

SLPs

Grammar form

More general rules: $X_i \to u X_j v X_k w, \quad j, k < i.$

Lemma

There are $|G| + 4n$ different maximal lengths of blocks in G.

Proof.

- blocks contained in explicit words: assign to explicit letters
- blocks not contained in explicit words: at most 4 per rule $\square$

SLPs

Grammar form

More general rules: $X_i \to uX_jvX_kw, \quad j,k < i.$

Lemma

There are $|G| + 4n$ different maximal lengths of blocks in G.

Proof.

- blocks contained in explicit words: assign to explicit letters
- blocks not contained in explicit words: at most 4 per rule $\square$

Lemma

There are $|G| + 4n$ different pairs of letters in G.

Blocks compression

Compression of a

Blocks compression

Compression of a

- $X_1 \to baaba$, $X_2 \to aaX_1baX_1baa$

Blocks compression

Compression of a

- $X_1 \to baaba$, $X_2 \to aaX_1baX_1baa$ (no problem)

Blocks compression

Compression of a

- $X_1 \to baaba$, $X_2 \to aaX_1baX_1baa$ (no problem)
- $X_1 \to a$, $X_2 \to aX_1aX_1a$

Blocks compression

Compression of a

- $X_1 \to baaba$, $X_2 \to aaX_1baX_1baa$ (no problem)
- $X_1 \to a$, $X_2 \to aX_1aX_1a$ (problem)

Blocks compression

Compression of a

- $X_1 \to baaba$, $X_2 \to aaX_1baX_1baa$ (no problem)
- $X_1 \to a$, $X_2 \to aX_1aX_1a$ (problem)
- $X_1 \to abaaba$, $X_2 \to aX_1aX_1a$

Blocks compression

Compression of a

- $X_1 \to baaba$, $X_2 \to aaX_1baX_1baa$ (no problem)
- $X_1 \to a$, $X_2 \to aX_1aX_1a$ (problem)
- $X_1 \to abaaba$, $X_2 \to aX_1aX_1a$ (problem)

Blocks compression

Compression of a

- $X_1 \to baaba$, $X_2 \to aaX_1baX_1baa$ (no problem)
- $X_1 \to a$, $X_2 \to aX_1aX_1a$ (problem)
- $X_1 \to abaaba$, $X_2 \to aX_1aX_1a$ (problem)

Definition (Crossing block)

a has a crossing block if some of its maximal blocks is contained in X_i but not in explicit words in X_i's rule.

Blocks compression

Compression of a

- $X_1 \to baaba$, $X_2 \to aaX_1baX_1baa$ (no problem)
- $X_1 \to a$, $X_2 \to aX_1aX_1a$ (problem)
- $X_1 \to abaaba$, $X_2 \to aX_1aX_1a$ (problem)

Definition (Crossing block)

a has a crossing block if some of its maximal blocks is contained in X_i but not in explicit words in X_i's rule.

When a has no crossing block

1: **for** all maximal blocks a^ℓ of a **do**
2: let $a_\ell \in \Sigma$ be an unused letter
3: replace each explicit maximal a^ℓ in rules' bodies by a_ℓ

What about crossing blocks?

Idea

- change the rules
- when X_i defines $a^{\ell_i} w a^{r_i} \mapsto w$
- replace X_i in rules by $a^{\ell_i} w a^{r_i}$

What about crossing blocks?

Idea

- change the rules
- when X_i defines $a^{\ell_i} w a^{r_i} \mapsto w$
- replace X_i in rules by $a^{\ell_i} w a^{r_i}$

CutPrefSuff(a)

for $i \leftarrow 1$ to n **do**
    calculate and remove a-prefix a^{ℓ_i} and a-suffix a^{r_i} of X_i
    replace each X_i in rules bodies by $a^{\ell_i} X_i a^{r_i}$

What about crossing blocks?

Idea

- change the rules
- when X_i defines $a^{\ell_i} w a^{r_i} \mapsto w$
- replace X_i in rules by $a^{\ell_i} w a^{r_i}$

CutPrefSuff(a)

for $i \leftarrow 1$ to n **do**
    calculate and remove a-prefix a^{ℓ_i} and a-suffix a^{r_i} of X_i
    replace each X_i in rules bodies by $a^{\ell_i} X_i a^{r_i}$

Lemma

After CutPrefSuff(a) *letter a has no crossing block.*

What about crossing blocks?

Idea

- change the rules
- when X_i defines $a^{\ell_i} w a^{r_i} \mapsto w$
- replace X_i in rules by $a^{\ell_i} w a^{r_i}$

CutPrefSuff(a)

1: **for** $i \leftarrow 1$ to n **do**
2: calculate and remove a-prefix a^{ℓ_i} and a-suffix a^{r_i} of X_i
3: replace each X_i in rules bodies by $a^{\ell_i} X_i a^{r_i}$

Lemma

After CutPrefSuff(a) *letter a has no crossing block.*

So a's blocks can be easily compressed.

What about crossing blocks?

Idea

- change the rules
- when X_i defines $a^{\ell_i} w a^{r_i} \mapsto w$
- replace X_i in rules by $a^{\ell_i} w a^{r_i}$

CutPrefSuff(a)

1: **for** $i \leftarrow 1$ to n **do**
2: calculate and remove a-prefix a^{ℓ_i} and a-suffix a^{r_i} of X_i
3: replace each X_i in rules bodies by $a^{\ell_i} X_i a^{r_i}$

Lemma

After CutPrefSuff(a) *letter a has no crossing block.*

So a's blocks can be easily compressed.

Parallelly for many letters!

What about crossing blocks?

Idea

- change the rules
- when X_i defines $a^{\ell_i} w b^{r_i} \mapsto w$
- replace X_i in rules by $a^{\ell_i} w b^{r_i}$

CutPrefSuff

1: **for** $i \leftarrow 1 \rightarrow n$ **do**
2: let X_i begin with a and end with b
3: calculate and remove a-prefix a^{ℓ} and b-suffix b^{r} of X_i
4: replace each X_i in rules bodies by $a^{\ell} X_i b^{r}$

Lemma

After CutPrefSuff *no letter has a crossing block.*

So all blocks can be easily compressed.

Pair compression

$X_1 \to ababcab$, $X_2 \to abcbX_1abX_1a$

Pair compression

$X_1 \to ababcab$, $X_2 \to abcbX_1abX_1a$

- compression of ab: easy

Pair compression

$X_1 \to ababcab$, $X_2 \to abcbX_1abX_1a$

- compression of ab: easy
- compression of ba: problem

Pair compression

$X_1 \to ababcab$, $X_2 \to abcbX_1abX_1a$

- compression of ab: easy
- compression of ba: problem
- pairs may overlap (problem: sequentially, not parallely)

Crossing pairs

When ab has a 'crossing' appearance: aX_i or $X_i b$

- X_i defines $bw \mapsto w$, replace X_i by bX_i
- symmetrically for ending a

Crossing pairs

When ab has a 'crossing' appearance: aX_i or $X_i b$

- X_i defines $bw \mapsto w$, replace X_i by bX_i
- symmetrically for ending a

LeftPop(b)

1: **for** i=1 to n **do**
2: **if** the first symbol in $X_i \to \alpha$ is b **then**
3: remove this b
4: replace X_i in productions by bX_i

Lemma

After LeftPop(b) *and* RightPop(a) *the* ab *is no longer crossing.*

Crossing pairs

When ab has a 'crossing' appearance: aX_i or $X_i b$

- X_i defines $bw \mapsto w$, replace X_i by bX_i
- symmetrically for ending a

LeftPop(b)

1: **for** i=1 to n **do**
2: **if** the first symbol in $X_i \to \alpha$ is b **then**
3: remove this b
4: replace X_i in productions by bX_i

Lemma

After LeftPop(b) *and* RightPop(a) *the ab is no longer crossing.*

Can be done in parallel!

Crossing pairs

When $ab \in \Sigma_1\Sigma_2$ has a crossing appearance: aX_i or $X_i b$

- X_i defines $bw \mapsto w$, replace X_i by aX_i
- symmetrically for ending a

LeftPop

1: **for** i=1 to n **do**
2: **if** the first symbol in $X_i \to \alpha$ is $b \in \Sigma_2$ **then**
3: remove this b
4: replace X_i in productions by bX_i

Lemma

After LeftPop *and* RightPop *the pairs* $\Sigma_1\Sigma_2$ *are no longer crossing.*

Running time

- Blocks compression: $\mathcal{O}(|G|)$ time
- non-crossing pairs: $\mathcal{O}(|G|)$ time
- crossing pairs: $\mathcal{O}(n+m)$ time per partition (Σ_1, Σ_2)

Running time

- Blocks compression: $\mathcal{O}(|G|)$ time
- non-crossing pairs: $\mathcal{O}(|G|)$ time
- crossing pairs: $\mathcal{O}(n+m)$ time per partition (Σ_1, Σ_2)

Lemma

There are $\mathcal{O}(n+m)$ crossing pairs.

Running time

- Blocks compression: $\mathcal{O}(|G|)$ time
- non-crossing pairs: $\mathcal{O}(|G|)$ time
- crossing pairs: $\mathcal{O}(n+m)$ time per partition (Σ_1, Σ_2)

Lemma

There are $\mathcal{O}(n+m)$ crossing pairs.

- crossing pairs: $\mathcal{O}((n+m)^2)$ time.

Running time

- Blocks compression: $\mathcal{O}(|G|)$ time
- non-crossing pairs: $\mathcal{O}(|G|)$ time
- crossing pairs: $\mathcal{O}(n+m)$ time per partition (Σ_1, Σ_2)

Lemma

There are $\mathcal{O}(n+m)$ crossing pairs.

- crossing pairs: $\mathcal{O}((n+m)^2)$ time.

Running time

Running time: $\mathcal{O}(|G| + (n+m)^2)$.

Shortening of the string

- consider pair ab in the text
- if $a = b$: it is compressed
- if $a \neq b$: it is compressed unless a or b was compressed already
- consider four consecutive symbols: something in them is compressed
- text compresses by a constant factor in each phase
- $\mathcal{O}(|\log M|)$ phases

Overall running time and grammar size

Grammar size

- In each phase size of grammar increases by $\mathcal{O}((n+m)^2)$
  - CutPrefSuff
  - LeftPop, RightPop
- shortening G: the same analysis as for pattern
  - shortens by a constant factor in a phase
- G is $\mathcal{O}((n+m)^2)$
- Running time is $\mathcal{O}((n+m)^2 \log M)$
- Can be reduced to $\mathcal{O}((n+m) \log M)$

Turning to the pattern matching

Problem with the ends

- text: *abababab*, pattern *baba*, compression of *ab*
- text: *abababab*, pattern *aba*, compression of *ab*
- text: *aaaaaaaa*, pattern *aaa*, compression of *a* blocks

Turning to the pattern matching

Problem with the ends

- text: *abababab*, pattern *baba*, compression of *ab*
- text: *abababab*, pattern *aba*, compression of *ab*
- text: *aaaaaaaa*, pattern *aaa*, compression of *a* blocks

Fixing the ends

- Compress the starting and ending pair, if possible (so *ba* in the first case)
- not possible, when the first and last letter is the same, say a
- replace leading a by a_L, ending by a_R
- spawn a into $a_R a_L$

- Questions?
- Other applications?