Fully compressed pattern matching by recompression

Artur Jeż
University of Wrocław

9 VII 2012
Definition (SLP: Straight Line Programme)

CFG generating exactly one word

\[ X_i \rightarrow X_jX_k \text{ or } X_i \rightarrow a \]
Definition (SLP: Straight Line Programme)

CFG generating exactly one word

\[ X_i \rightarrow X_jX_k \text{ or } X_i \rightarrow a \]

Example

\[ X_0 = a, \ X_1 = b, \ X_{n+1} = X_{n-1}X_{n-2} \]

\[ a, \ b, \ ba, \ bab, \ babba, \ babbababb, \ldots \]
SLP

Definition (SLP: Straight Line Programme)

CFG generating exactly one word

\[ X_i \rightarrow X_j X_k \text{ or } X_i \rightarrow a \]

Example

\[ X_0 = a, \ X_1 = b, \ X_{n+1} = X_{n-1}X_{n-2} \]

\[ a, \ b, \ ba, \ bab, \ babba, \ babbababb, \ldots \]

Relations to LZ and LZW

LZW rules \( X_i \rightarrow aX_j \), text is \( X_1X_2X_3\ldots \)

LZ LZ to SLP: from \( n \) to \( \mathcal{O}(n \log(N/n)) \)
**Definition (SLP: Straight Line Programme)**

CFG generating exactly one word

\[ X_i \rightarrow X_jX_k \text{ or } X_i \rightarrow a \]

**Example**

\[ X_0 = a, \ X_1 = b, \ X_{n+1} = X_{n-1}X_{n-2} \]

\[ a, b, ba, bab, babba, babbababb, \ldots \]

**Relations to LZ and LZW**

LZW rules \( X_i \rightarrow aX_j \), text is \( X_1X_2X_3\ldots \)

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...)

---

Artur Jeż

FCPM by recompression

9 VII 2012 2 / 18
This talk

Definition (CPM, FCPM)

Compressed pattern matching: text is compressed, pattern not.
Fully Compressed pattern matching: both text and pattern are compressed.
Definition (CPM, FCPM)
Compressed pattern matching: text is compressed, pattern not.
Fully Compressed pattern matching: both text and pattern are compressed.

Results
An $O((n + m) \log M)$ algorithm for FCPM for SLP.
(Previously: $O(nm^2)$, [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 $O((n + m) \log M)$ algorithm for FCPM for SLP.
(Previously: $O(nm^2)$, [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 \([\in \text{NP}]\)
- Word equations [alternative PSPACE algorithm]
- Fully Compressed Pattern Matching [SLPs, LZ, \(O((n + m) \log M \log(n + m))\)]
- construction of a grammar for a string [alternative \(\log(N/n)\) approximation algorithm]
- other?
Equality of strings

How to test equality of strings?

\[ \begin{align*}
a & a a a b b a b c a b a b b a b c b b b a \\
a & a a a b b a b c a b a b b a b c b b b a \\
\end{align*} \]
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  b  a
```

```
a  a  a  b  a  b  c  a  b  a  b  b  a  b  c  b  b  a
```
Example

Equality of strings

How to test equality of strings?

\[ a_3 \ b \ a \ b \ c \ a \ b \ a \ b \ b \ a \ b \ c \ b \ b \ a \]

\[ a_3 \ b \ a \ b \ c \ a \ b \ a \ b \ b \ a \ b \ c \ b \ b \ a \]
Example

Equality of strings

How to test equality of strings?

\[ a_3 \ b \ a \ b \ c \ a \ b \ a \ b_2 \ a \ b \ c \ b \ a \ b \]

\[ a_3 \ b \ a \ b \ c \ a \ b \ a \ b_2 \ a \ b \ c \ b \ a \ b \]
Example

Equality of strings

How to test equality of strings?

\[ a_3 \ b \ d \ c \ d \ a \ b_2 \ d \ c \ b \ a \]

\[ a_3 \ b \ d \ c \ d \ a \ b_2 \ d \ c \ b \ a \]
Example

Equality of strings

How to test equality of strings?

\[ a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e \]

\[ a_3 \quad b \quad d \quad c \quad d \quad a \quad b_2 \quad d \quad c \quad e \]
Example

Equality of strings

How to test equality of strings?

\[ a_3 \ b \ d \ c \ d \ a \ b_2 \ d \ c \ e \]

\[ a_3 \ b \ d \ c \ d \ a \ b_2 \ d \ c \ e \]
Example

Equality of strings

How to test equality of strings?

\[ a_3 \ b \ d \ c \ d \ a \ b_2 \ d \ c \ e \]

\[ a_3 \ b \ d \ c \ d \ a \ b_2 \ d \ c \ e \]

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

\[
\begin{align*}
L & \leftarrow \text{list of letters,} \\
P & \leftarrow \text{list of pairs of letters}
\end{align*}
\]

for every letter \(a \in L\) do

replace \((\text{maximal}) \ a \ell\) with \(a \ell\)

for every pair of letter \(ab \in P\) do

replace \(ab\) with \(c\)

It will shorten the strings by constant factor.

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

Artur Jeż

FCPM by recompression

9 VII 2012 7 / 18
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
- \textbf{for} every letter \( a \in L \) \textbf{do}
  - replace (maximal) blocks \( a^\ell \) with \( a_\ell \)

It will shorten the strings by constant factor.
Loop, while nontrivial. (\( O(\log M) \) iterations).

Artur Jeż
FCPM by recompression
Formalisation

In one phase

- \( L \leftarrow \text{list of letters}, \ P \leftarrow \text{list of pairs of letters} \)
- \( \textbf{for} \ \text{every letter } a \in L \ \textbf{do} \)
  - replace (maximal) blocks \( a^\ell \) with \( a_\ell \)
- \( \textbf{for} \ \text{every pair of letter } ab \in P \ \textbf{do} \)
  - replace pairs \( ab \) with \( c \)
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 $a^\ell$ with $a_\ell$
- 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

- \( L \leftarrow \text{list of letters}, \ P \leftarrow \text{list of pairs of letters} \)
- \( \textbf{for} \ \text{every letter} \ a \in L \ \textbf{do} \)
  - replace (maximal) blocks \( a^{\ell} \) with \( a_{\ell} \)
- \( \textbf{for} \ \text{every pair of letter} \ ab \in P \ \textbf{do} \)
  - replace pairs \( ab \) with \( c \)

It will shorten the strings by constant factor.

Loop, while nontrivial.
(\( O(\log M) \) iterations).
SLPs

Grammar form

More general rules: \[ X_i \rightarrow uX_jvX_kw, \quad j, k < i. \]
SLPs

Grammar form

More general rules: \( X_i \rightarrow uX_j vX_k w, \ j, k < i. \)

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
SLPs

Grammar form

More general rules: \[ X_i \rightarrow uX_jvX_kw, \quad j, k < i. \]

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

Lemma

There are \(|G| + 4n\) different pairs of letters in \(G\).
Blocks compression

Compression of $a$

1: for all maximal blocks $a^\ell$ of $a$
2: let $a^\ell \in \Sigma$ be an unused letter
3: replace each explicit maximal $a^\ell$ in rules’ bodies by $a^\ell$
Blocks compression

Compression of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1 baX_1 baa$
Blocks compression

Compression of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1 baX_1 baa$ (no problem)
Blocks compression

Compression of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1baX_1baa$ (no problem)
- $X_1 \rightarrow a$, $X_2 \rightarrow aX_1aX_1a$
Blocks compression

Compression of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1 baX_1 baa$ (no problem)
- $X_1 \rightarrow a$, $X_2 \rightarrow aX_1 aX_1 a$ (problem)
Blocks compression

Compression of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1 baX_1 baa$ (no problem)
- $X_1 \rightarrow a$, $X_2 \rightarrow aX_1 aX_1 a$ (problem)
- $X_1 \rightarrow abaaba$, $X_2 \rightarrow aX_1 aX_1 a$
Blocks compression

**Compression of \( a \)**

- \( X_1 \rightarrow baaba, \ X_2 \rightarrow aaX_1 baX_1 baa \) (no problem)
- \( X_1 \rightarrow a, \ X_2 \rightarrow aX_1 aX_1 a \) (problem)
- \( X_1 \rightarrow abaaba, \ X_2 \rightarrow aX_1 aX_1 a \) (problem)
Blocks compression

Compression of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1 baX_1 baa$ (no problem)
- $X_1 \rightarrow a$, $X_2 \rightarrow aX_1 aX_1 a$ (problem)
- $X_1 \rightarrow abaaba$, $X_2 \rightarrow aX_1 aX_1 a$ (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**

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1baX_1baa$ (no problem)
- $X_1 \rightarrow a$, $X_2 \rightarrow aX_1aX_1a$ (problem)
- $X_1 \rightarrow abaaba$, $X_2 \rightarrow aX_1aX_1a$ (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 $a^\ell$ of *a* **do**
2: let $a^\ell \in \Sigma$ be an unused letter
3: replace each explicit maximal $a^\ell$ in rules’ bodies by $a^\ell$
What about crossing blocks?

Idea

- change the rules
- when $X_i$ defines $a^{\ell_i} w a^{r_i} \rightarrow w$
- 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)

1: for $i \leftarrow 1$ to $n$ do
2: calculate and remove $a$-prefix $a^{\ell_i}$ 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 $a^l_i w a^r_i \rightarrow w$
- replace $X_i$ in rules by $a^l_i w a^r_i$

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 $a^l_i X_i a^r_i$

Lemma

After CutPrefSuff$(a)$ letter $a$ has no crossing block.
What about crossing blocks?

**Idea**

- change the rules
- when $X_i$ defines $a^\ell_i w a^r_i \rightarrow w$
- 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 $a^\ell_i$ 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 $a^l_i w a^r_i \rightarrow w$
- replace $X_i$ in rules by $a^l_i w a^r_i$

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 $a^l_i X_i a^r_i$

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 $a^i w b^r_i \mapsto w$
- replace $X_i$ in rules by $a^i w b^r_i$

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 $a^l X_i b^r$

Lemma
After CutPrefSuff no letter has a crossing block.

So all blocks can be easily compressed.
Pair compression

\[ X_1 \rightarrow ababcab, \ X_2 \rightarrow abcbX_1abX_1a \]
Pair compression

$X_1 \rightarrow ababcab$, $X_2 \rightarrow abcbX_1 abX_1 a$

- compression of $ab$: easy
Pair compression

\[ X_1 \rightarrow ababca b, \ X_2 \rightarrow abcbX_1abX_1a \]

- compression of \( ab \): easy
- compression of \( ba \): problem
Pair compression

\[ X_1 \rightarrow ababcab, \ X_2 \rightarrow abcbX_1abX_1a \]

- 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 $X_ib$

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

Lemma

After $\text{LeftPop}(b)$ and $\text{RightPop}(a)$ the $ab$ is no longer crossing.

Can be done in parallel!
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: \textbf{for} \( i=1 \) to \( n \) \textbf{do}
2: \hspace{1em} \textbf{if} the first symbol in \( X_i \rightarrow \alpha \) is \( b \) \textbf{then}
3: \hspace{2em} remove this \( b \)
4: \hspace{2em} 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 \( X_ib \)
- \( X_i \) 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 \( X_i \rightarrow \alpha \) 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 $X_ib$
- $X_i$ 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 $X_i \rightarrow \alpha$ 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: $O(|G|)$ time
- non-crossing pairs: $O(|G|)$ time
- crossing pairs: $O(n + m)$ time per partition $(\Sigma_1, \Sigma_2)$
Running time

- Blocks compression: $O(|G|)$ time
- non-crossing pairs: $O(|G|)$ time
- crossing pairs: $O(n + m)$ time per partition ($\Sigma_1, \Sigma_2$)

Lemma

There are $O(n + m)$ 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 ($\Sigma_1, \Sigma_2$)

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 $(\Sigma_1, \Sigma_2)$

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
- $O(|\log M|)$ phases
Grammar size

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

Problem with the ends

- text: \textit{abababab}, pattern \textit{baba}, compression of \textit{ab}
- text: \textit{abababab}, pattern \textit{aba}, compression of \textit{ab}
- text: \textit{aaaaaaaaa}, pattern \textit{aaa}, compression of \textit{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: \(aaaaaaaaa\), 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 \(a_Ra_L\)
Questions?

Other applications?