Fully compressed pattern matching by recompression

Artur Jeź
University of Wrocław

9 VII 2012
Definition (SLP: Straight Line Programme)

CFG generating exactly one word

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

Definition (SLP: Straight Line Programme)

CFG generating exactly one word

\( X_i \rightarrow X_j X_k \) 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

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

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

\begin{align*}
\text{LZW rules } & X_i \rightarrow aX_j, \text{ text is } X_1X_2X_3\ldots \\
\text{LZ } & \text{ LZ to SLP: from } n \text{ to } \mathcal{O}(n \log(N/n))
\end{align*}
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 \( X_1 X_2 X_3 \ldots \)

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...)
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 $O((n + m) \log M)$ 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 [$\in$ 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?
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?

\[
\begin{align*}
\text{a a a b a b c a b a b b a b c b b a} \\
\text{a a a b a b c a b a b b a b c b b a}
\end{align*}
\]
Equality of strings

How to test equality of strings?

\[
\begin{align*}
a_3 & \quad b \quad a \quad b \quad c \quad a \quad b \quad a \quad b \quad b \quad a \quad b \quad c \quad b \quad a \\
a_3 & \quad b \quad a \quad b \quad c \quad a \quad b \quad a \quad b \quad b \quad a \quad b \quad c \quad b \quad a
\end{align*}
\]
Example

Equality of strings

How to test equality of strings?

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

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

Equality of strings

How to test equality of strings?

\[ \begin{align*}
  a_3 & \quad b & \quad d & \quad c & \quad d & \quad a & \quad b_2 & \quad d & \quad c & \quad b & \quad a \\
  a_3 & \quad b & \quad d & \quad c & \quad d & \quad a & \quad b_2 & \quad d & \quad c & \quad b & \quad a
\end{align*} \]
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?

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

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

Equality of strings
How to test equality of strings?

\[
\begin{align*}
& a_3 b d c d a b_2 d c e \\
& a_3 b d c d a b_2 d c e
\end{align*}
\]

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

\[
L \leftarrow \text{list of letters,} \\
\text{P} \leftarrow \text{list of pairs of letters} \\
\text{for every letter} \ a \ \in \ L \\
\text{do} \\
\text{replace} \ \text{maximal} \ \text{blocks} \ a \ \ell \ \text{with} \ a \ \ell \\
\text{for every pair of letters} \ ab \ \in \ P \\
\text{do} \\
\text{replace pairs} \ ab \ \text{with} \ c
\]

It will shorten the strings by constant factor.

Loop, while nontrivial. \( (O(\log M) \text{ iterations}). \)
In one phase

- $L \leftarrow$ list of letters, $P \leftarrow$ list of pairs of letters
In one phase

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

It will shorten the strings by constant factor.

Loop, while nontrivial. ($O(\log M)$ iterations).
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 $a^\ell$ with $a_\ell$
- 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 $a^\ell$ 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

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

It will shorten the strings by constant factor.

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

Grammar form

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

Grammar form
More general rules: \( X_i \rightarrow uX_jvX_kw, \quad j, k < i \).

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
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 \(G\).
Compresssion of $a$

$\text{Definition (Crossing block)}$

$a$ 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 $a$

2: let $a_\ell \in \Sigma$ be an unused letter

3: replace each explicit maximal $a_\ell$ in rules' bodies by $a_\ell$

Artur Jeż

FCPM by recompression

9 VII 2012 9 / 17
Blocks compression

Compression of $a$

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

Compression of $a$

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

Compression of $a$

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

Definition (Crossing block)

A letter $a$ 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 $a^\ell$
Blocks compression

Compression of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1baX_1bba$ (no problem)
- $X_1 \rightarrow a$, $X_2 \rightarrow aX_1aX_1a$ (problem)
Blocks compression

Compresssion of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1bX_1bba$ (no problem)
- $X_1 \rightarrow a$, $X_2 \rightarrow aX_1aX_1a$ (problem)
- $X_1 \rightarrow abaaba$, $X_2 \rightarrow aX_1aX_1a$
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)
- $X_1 \rightarrow abaaba$, $X_2 \rightarrow aX_1 aX_1 a$ (problem)
Blocks compression

Compression of $a$

- $X_1 \rightarrow baaba$, $X_2 \rightarrow aaX_1baX_1baa$ (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.
**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.

### When $a$ has no crossing block

1. **for** all maximal blocks $a^l$ of $a$ **do**
2. let $a^l \in \Sigma$ 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 $a^{\ell_i}wa^{r_i} \rightarrow w$
- replace $X_i$ in rules by $a^{\ell_i}wa^{r_i}$

Lemma

After $\text{CutPrefSuff}(a)$ 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 $a^l_i X_i a^r_i$
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.
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.
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 $a^{ℓ_i}X_ia^{r_i}$

**Lemma**

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

So $a$’s blocks can be easily compressed.

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

Lemma

After CutPrefSuff \textit{no letter has a crossing block.}

So \textbf{all blocks} can be easily compressed.
Pair compression

\( X_1 \rightarrow ababcab, \ X_2 \rightarrow abcbX_1abX_1a \)
Pair compression

\( X_1 \rightarrow ababcab, \ X_2 \rightarrow abcbX_1abX_1a \)

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

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

- 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 LeftPop(\(b\)) and RightPop(\(a\)) the \(ab\) is no longer crossing.

Can be done in parallel!
Crossing pairs

When \( ab \) has a ‘crossing’ appearance: \( aX_i \) or \( X_ib \)

- \( X_i \) defines \( bw \leftrightarrow w \), replace \( X_i \) by \( bX_i \)
- symmetrically for ending \( a \)

LeftPop(\( b \))

1: \textbf{for} \( i=1 \) to \( n \) \textbf{do}
2: \textbf{if} the first symbol in \( X_i \rightarrow \alpha \) is \( b \) \textbf{then}
3: \hspace{1em} remove this \( b \)
4: \hspace{1em} 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. \textbf{for} \( i=1 \) to \( n \) \textbf{do}
2. \hspace{1em} \textbf{if} the first symbol in \( X_i \rightarrow \alpha \) is \( b \in \Sigma_2 \) \textbf{then}
3. \hspace{2em} remove this \( b \)
4. \hspace{2em} replace \( X_i \) in productions by \( bX_i \)

Lemma

After LeftPop and RightPop the \textit{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: $O(|G|)$ 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.

- crossing pairs: $O((n + m)^2)$ time.
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.

- crossing pairs: $O((n + m)^2)$ time.

Running time

Running time: $O(|G| + (n + m)^2)$. 
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
Overall running time and grammar size

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{aaaaaaaa}, pattern \textit{aaa}, compression of \textit{a} blocks
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}

Fixing the ends

- Compress the starting and ending pair, if possible (so \textit{ba} in the first case)
- not possible, when the first and last letter is the same, say \textit{a}
- replace leading \textit{a} by \textit{a}_L, ending by \textit{a}_R
- spawn \textit{a} into \textit{a}_Ra_L
- Questions?
- Other applications?