Smallest grammar by recompression

Artur Jeż

Max Planck Institute for Informatics

17.06.2013

Grammar based-compression

Represent *w* as a CFG generating it.

Grammar based-compression

Represent w as a CFG generating it.

Advantages

- it is usually small (at most quadratic vs. LZ)
- compression is fast
- it is exponential on good data

Grammar based-compression

Represent w as a CFG generating it.

Advantages

- it is usually small (at most quadratic vs. LZ)
- compression is fast
- it is exponential on good data
- extracts hierarchical structure
- it is easy to work on

Grammar based-compression

Represent *w* as a CFG generating it.

Advantages

- it is usually small (at most quadratic vs. LZ)
- compression is fast
- it is exponential on good data
- extracts hierarchical structure
- it is easy to work on
- related to LZW and LZ

Smallest grammar

Problem

Given w return smallest CFG G_w such that $L(G_w) = w$.

Smallest grammar

Problem

Given w return smallest CFG G_w such that $L(G_w) = w$.

With $\mathcal{O}(1)$ increase in size, this is an SLP.

Definition (SLP: Straight Line Programme)

CFG with

- ordered nonterminals $X_1, X_2, \ldots$
- Chomsky normal form
- for $X_i \to X_j X_k$ we have $j, k < i$

What is known

Best approximation ratio

$\mathcal{O}(\log(n/g))$, where g is the size of the optimal grammar.

What is known

Best approximation ratio

$\mathcal{O}(\log(n/g))$, where g is the size of the optimal grammar.

- Rytter
  - represent w as LZ, size $\ell \leq g$
  - translation of LZ into SLP, size $\mathcal{O}(\ell \log(n/\ell)) \leq \mathcal{O}(g \log(n/g))$
  - the intermediate grammar is balanced (AVL-type condition)

What is known

Best approximation ratio

$\mathcal{O}(\log(n/g))$, where g is the size of the optimal grammar.

- Rytter
  - represent w as LZ, size $\ell \leq g$
  - translation of LZ into SLP, size $\mathcal{O}(\ell \log(n/\ell)) \leq \mathcal{O}(g \log(n/g))$
  - the intermediate grammar is balanced (AVL-type condition)
- Charikar et al.:
  - similar as Rytter
  - different balance criterion (length of word)

What is known

Best approximation ratio

$\mathcal{O}(\log(n/g))$, where g is the size of the optimal grammar.

- Rytter
  - represent w as LZ, size $\ell \leq g$
  - translation of LZ into SLP, size $\mathcal{O}(\ell \log(n/\ell)) \leq \mathcal{O}(g \log(n/g))$
  - the intermediate grammar is balanced (AVL-type condition)
- Charikar et al.:
  - similar as Rytter
  - different balance criterion (length of word)
- Sakamoto
  - local replacement rules (plus a global partition): pairs and blocks
  - analysis vs LZ

What is known

Best approximation ratio

$\mathcal{O}(\log(n/g))$, where g is the size of the optimal grammar.

- Rytter
  - represent w as LZ, size $\ell \leq g$
  - translation of LZ into SLP, size $\mathcal{O}(\ell \log(n/\ell)) \leq \mathcal{O}(g \log(n/g))$
  - the intermediate grammar is balanced (AVL-type condition)
- Charikar et al.:
  - similar as Rytter
  - different balance criterion (length of word)
- Sakamoto
  - local replacement rules (plus a global partition): pairs and blocks
  - analysis vs LZ

Linear time.

This talk

Very simple linear-time algorithm, $\mathcal{O}(\log(n/g))$ approximation.

This talk

Very simple linear-time algorithm, $\mathcal{O}(\log(n/g))$ approximation.

- analysis in the recompression framework, vs. SLP
  - very robust
  - good: easier to show better approximation?
  - bad: might be in fact larger

This talk

Very simple linear-time algorithm, $\mathcal{O}(\log(n/g))$ approximation.

- analysis in the recompression framework, vs. SLP
  - very robust
  - good: easier to show better approximation?
  - bad: might be in fact larger
- not balanced
  - good: easier to show approximation?
  - bad: worse for further processing

This talk

Very simple linear-time algorithm, $\mathcal{O}(\log(n/g))$ approximation.

- analysis in the recompression framework, vs. SLP
  - very robust
  - good: easier to show better approximation?
  - bad: might be in fact larger
- not balanced
  - good: easier to show approximation?
  - bad: worse for further processing
- height $\mathcal{O}(\log n)$, when a^ℓ has height 1

This talk

Very simple linear-time algorithm, $\mathcal{O}(\log(n/g))$ approximation.

- analysis in the recompression framework, vs. SLP
  - very robust
  - good: easier to show better approximation?
  - bad: might be in fact larger
- not balanced
  - good: easier to show approximation?
  - bad: worse for further processing
- height $\mathcal{O}(\log n)$, when a^ℓ has height 1

Algorithm similar to Sakamoto, different analysis.

Example

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

Example

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

Example

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

$$a_3 \rightarrow a^3$$

Example

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

$$a_3 \rightarrow a^3, b_2 \rightarrow b^2$$

Example

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

$$a_3 \to a^3, b_2 \to b^2, d \to ab$$

Example

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

$$a_3 \to a^3, b_2 \to b^2, d \to ab, e \to ba$$

Example

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

$$a_3 \to a^3, b_2 \to b^2, d \to ab, e \to ba$$

Example

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

$$a_3 \to a^3, b_2 \to b^2, d \to ab, e \to ba$$

Intuition

- Phases: compress only pairs and block from the beginning of a phase.
- Treat nonterminals as letters.
- To speed up, we make some pair compression simultaneously (partition Σ to Σ_ℓ, Σ_r, pairs from $\Sigma_\ell\Sigma_r$)

Algorithm

1: **while** $|T| > 1$ **do**

Algorithm

```
while |T| > 1 do
    L ← list of letters in T
    for each a ∈ L do                              ▷ Blocks compression
        compress maximal blocks of a               ▷ O(|T|)
```

Algorithm

while $|T| > 1$ **do**
  $L \leftarrow$ list of letters in T
  for each $a \in L$ **do** $\triangleright$ Blocks compression
    compress maximal blocks of a $\triangleright \mathcal{O}(|T|)$
  $P \leftarrow$ list of pairs
  find partition of Σ into Σ_ℓ and Σ_r
    $\triangleright$ Try to maximize the occurrences from $\Sigma_\ell \Sigma_r$ in T.

Algorithm

while $|T| > 1$ **do**
  $L \leftarrow$ list of letters in T
  for each $a \in L$ **do** ▷ Blocks compression
    compress maximal blocks of a ▷ $\mathcal{O}(|T|)$
  $P \leftarrow$ list of pairs
  find partition of Σ into Σ_ℓ and Σ_r
    ▷ Try to maximize the occurrences from $\Sigma_\ell \Sigma_r$ in T.
  for $ab \in P \cap \Sigma_\ell \Sigma_r$ **do** ▷ These pairs do not overlap
    compress pair ab ▷ Pair compression

Algorithm

while $|T| > 1$ **do**
  $L \leftarrow$ list of letters in T
  for each $a \in L$ **do** ▷ Blocks compression
    compress maximal blocks of a ▷ $\mathcal{O}(|T|)$
  $P \leftarrow$ list of pairs
  find partition of Σ into Σ_ℓ and Σ_r
    ▷ Try to maximize the occurrences from $\Sigma_\ell\Sigma_r$ in T.
  for $ab \in P \cap \Sigma_\ell\Sigma_r$ **do** ▷ These pairs do not overlap
    compress pair ab ▷ Pair compression
return the constructed grammar

Partition

$1/4$ appearances covered

A partition $\Sigma_\ell \Sigma_r$ such that $1/4$ of pairs is covered.

Partition

1/4 appearances covered

A partition $\Sigma_\ell \Sigma_r$ such that $1/4$ of pairs is covered.

- After block compression *aa* does not appear.
- Random partition: $1/4$ pairs can be covered.
- derandomise (expected value)
- we need number of appearances of *ab*: RadixSort
- $\mathcal{O}(|T|)$.

Size reduction

Size drop

- Consider set of two consecutive letters ab in T.
- For $1/4$ of them one letter is compressed in a phase.
- Length drops by a constant factor.

Size reduction

Size drop

- Consider set of two consecutive letters *ab* in T.
- For $1/4$ of them one letter is compressed in a phase.
  - if $a = b$: it is compressed
- Length drops by a constant factor.

Size reduction

Size drop

- Consider set of two consecutive letters *ab* in T.
- For $1/4$ of them one letter is compressed in a phase.
  - if $a = b$: it is compressed
  - if $a \neq b$: $1/4$ of those pairs is in $\Sigma_\ell \Sigma_r$
    When we consider *ab* we replace it, unless one letter was already replaced.
- Length drops by a constant factor.

Size reduction

Size drop

- Consider set of two consecutive letters *ab* in T.
- For $1/4$ of them one letter is compressed in a phase.
  - if $a = b$: it is compressed
  - if $a \neq b$: $1/4$ of those pairs is in $\Sigma_\ell \Sigma_r$
    When we consider *ab* we replace it, unless one letter was already replaced.
- Length drops by a constant factor.

Towards running time

It is enough to show that one round runs in $\mathcal{O}(|T|)$.

Running time

Partition

$\mathcal{O}(|T|)$ time.

Block compression

By RadixSort, $\mathcal{O}(|T|)$ time.

Pair compression

By RadixSort, $\mathcal{O}(|T|)$ time.

Number of nonterminals

Representation cost

Number of nonterminals

Representation cost

- when c replaces ab we add rule $c \to ab$, representation cost 1

Number of nonterminals

Representation cost

- when c replaces ab we add rule $c \to ab$, representation cost 1
- when $a^{\ell_1}, a^{\ell_2}, \dots, a^{\ell_k}$ are replaced with $a_{\ell_1}, a_{\ell_2}, \dots, a_{\ell_k}$ $(\ell_1 < \ell_2 \dots < \ell_k)$:

Number of nonterminals

Representation cost

- when c replaces ab we add rule $c \to ab$, representation cost 1
- when $a^{\ell_1}, a^{\ell_2}, \dots, a^{\ell_k}$ are replaced with $a_{\ell_1}, a_{\ell_2}, \dots, a_{\ell_k}$ $(\ell_1 < \ell_2 \dots < \ell_k)$:
  - first represent $a^{\ell_2 - \ell_1}, a^{\ell_3 - \ell_2}, \dots, a^{\ell_k - \ell_{k-1}}$ as $a_{\ell_2 - \ell_1}, a_{\ell_3 - \ell_2}, \dots, a_{\ell_k - \ell_{k-1}}$
  - do this by binary expansion
    (make new rules $a_2 \to aa$, $a_4 \to a_2 a_2$, $a_8 \to a_4 a_4, \dots$)

Number of nonterminals

Representation cost

- when c replaces ab we add rule $c \to ab$, representation cost 1
- when $a^{\ell_1}, a^{\ell_2}, \ldots, a^{\ell_k}$ are replaced with $a_{\ell_1}, a_{\ell_2}, \ldots, a_{\ell_k}$ $(\ell_1 < \ell_2 \ldots < \ell_k)$:
  - first represent $a^{\ell_2-\ell_1}, a^{\ell_3-\ell_2}, \ldots, a^{\ell_k-\ell_{k-1}}$ as $a_{\ell_2-\ell_1}, a_{\ell_3-\ell_2}, \ldots, a_{\ell_k-\ell_{k-1}}$
  - do this by binary expansion (make new rules $a_2 \to aa$, $a_4 \to a_2a_2$, $a_8 \to a_4a_4, \ldots$)
  - $a_{\ell_{i+1}} \to a_{\ell_{i+1}-\ell_i} a_{\ell_i}$

Number of nonterminals

Representation cost

- when c replaces ab we add rule $c \to ab$, representation cost 1
- when $a^{\ell_1}, a^{\ell_2}, \ldots, a^{\ell_k}$ are replaced with $a_{\ell_1}, a_{\ell_2}, \ldots, a_{\ell_k}$ ($\ell_1 < \ell_2 \ldots < \ell_k$):
  - first represent $a^{\ell_2-\ell_1}, a^{\ell_3-\ell_2}, \ldots, a^{\ell_k-\ell_{k-1}}$ as $a_{\ell_2-\ell_1}, a_{\ell_3-\ell_2}, \ldots, a_{\ell_k-\ell_{k-1}}$
  - do this by binary expansion (make new rules $a_2 \to aa$, $a_4 \to a_2a_2$, $a_8 \to a_4a_4, \ldots$)
  - $a_{\ell_{i+1}} \to a_{\ell_{i+1}-\ell_i} a_{\ell_i}$
  - representation cost

$$\mathcal{O}\Big(\sum_{i=1}^{k-1} \log(\ell_{i+1} - \ell_i)\Big)$$

Analysis outline

- We begin with a G generating T (mental experiment)
- in each moment we keep G generating the current T

Analysis outline

- We begin with a G generating T (mental experiment)
- in each moment we keep G generating the current T
  - we apply the compression to G
  - it is changed so that this can be done

Analysis outline

- We begin with a G generating T (mental experiment)
- in each moment we keep G generating the current T
  - we apply the compression to G
  - it is changed so that this can be done
- representation cost is calculated using G

Analysis outline

- We begin with a G generating T (mental experiment)
- in each moment we keep G generating the current T
  - we apply the compression to G
  - it is changed so that this can be done
- representation cost is calculated using G

- G is of more general form: $X_i \to uX_jvX_kw$
- explicit letters have credit
- representation cost is paid by released credit:

Analysis outline

- We begin with a G generating T (mental experiment)
- in each moment we keep G generating the current T
  - we apply the compression to G
  - it is changed so that this can be done
- representation cost is calculated using G

- G is of more general form: $X_i \to uX_jvX_kw$
- explicit letters have credit
- representation cost is paid by released credit:
  - ab is replaced by c
  - we need 1 representation cost
  - each ab in G is replaced with c, 1 credit is released

Analysis outline

- We begin with a *G* generating *T* (mental experiment)
- in each moment we keep *G* generating the current *T*
  - we apply the compression to *G*
  - it is changed so that this can be done
- representation cost is calculated using *G*

- *G* is of more general form: $X_i \to uX_jvX_kw$
- explicit letters have credit
- representation cost is paid by released credit:
  - *ab* is replaced by *c*
  - we need 1 representation cost
  - each *ab* in *G* is replaced with *c*, 1 credit is released
  - (bit more tricky for blocks)

Analysis outline

- We begin with a G generating T (mental experiment)
- in each moment we keep G generating the current T
  - we apply the compression to G
  - it is changed so that this can be done
- representation cost is calculated using G

- G is of more general form: $X_i \to uX_jvX_kw$
- explicit letters have credit
- representation cost is paid by released credit:
  - ab is replaced by c
  - we need 1 representation cost
  - each ab in G is replaced with c, 1 credit is released
  - (bit more tricky for blocks)
- we only need to count the number of created credit

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

Definition (Non-crossing pairs)

ab is non-crossing pair iff none of the below happens

- aX appears in a rule, X begins with b
- Xb appears in a rule, X ends with a

Pair compression

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

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

Definition (Non-crossing pairs)

ab is non-crossing pair iff none of the below happens

- aX appears in a rule, X begins with b
- Xb appears in a rule, X ends with a

When each pair from $\Sigma_\ell\Sigma_r$ is non-crossing,
replace all those pairs in G (no new credit).

Making pairs non-crossing

When *ab* has a crossing appearance: aX_i or X_ib

- X_i defines *bw*: change it to *w*, replace X_i by bX_i
- symmetrically for ending *a*

Making pairs non-crossing

When ab has a crossing appearance: aX_i or X_ib

- X_i defines bw: change it to w, replace X_i by bX_i
- symmetrically for ending a

LeftPop(b)

for $i \leftarrow 1 \, .. \, g-1$ **do**
  if the first symbol in $X_i \rightarrow \alpha$ is b **then**
    remove this b
    replace X_i in productions by bX_i

Lemma

After LeftPop(b) *and* RightPop(a) *the ab is non-crossing.*

Making pairs non-crossing

When ab has a crossing appearance: aX_i or X_ib

- X_i defines bw: change it to w, replace X_i by bX_i
- symmetrically for ending a

LeftPop(b)

1: **for** $i \leftarrow 1 \,..\, g-1$ **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 non-crossing.*

- Can be done in parallel for all $ab \in \Sigma_\ell \Sigma_r$.

Making pairs non-crossing

When $ab \in \Sigma_\ell\Sigma_r$ has a crossing appearance: aX_i or X_ib

- X_i defines bw: change it to w, replace X_i by aX_i
- symmetrically for ending a

LeftPop

for $i \leftarrow 1 \,..\, g-1$ **do**
  if the first symbol in $X_i \to \alpha$ is $b \in \Sigma_r$ **then**
    remove this b
    replace X_i in productions by bX_i

Lemma

After LeftPop *and* RightPop *the pairs* $\Sigma_\ell\Sigma_r$ *are non-crossing.*

- Can be done in parallel for all $ab \in \Sigma_\ell\Sigma_r$.

Making pairs non-crossing

When $ab \in \Sigma_\ell \Sigma_r$ has a crossing appearance: aX_i or $X_i b$

- X_i defines bw: change it to w, replace X_i by aX_i
- symmetrically for ending a

LeftPop

for $i \leftarrow 1 \mathinner{.\,.} g-1$ **do**
    if the first symbol in $X_i \to \alpha$ is $b \in \Sigma_r$ **then**
        remove this b
        replace X_i in productions by bX_i

Lemma

After LeftPop *and* RightPop *the pairs* $\Sigma_\ell \Sigma_r$ *are non-crossing.*

- Can be done in parallel for all $ab \in \Sigma_\ell \Sigma_r$.
- Credit increases by $\mathcal{O}(g)$

Blocks & Wrap up

Idea

Similarly as pairs

- X_i defines $a^{\ell_i} w b^{r_i}$: change it to w
- replace X_i in rules by $a^{\ell_i} X_i b^{r_i}$

Blocks & Wrap up

Idea

Similarly as pairs

- X_i defines $a^{\ell_i} w b^{r_i}$: change it to w
- replace X_i in rules by $a^{\ell_i} X_i b^{r_i}$
- analysis: more tricky but works
- $\mathcal{O}(g)$

Blocks & Wrap up

Idea

Similarly as pairs

- X_i defines $a^{\ell_i} w b^{r_i}$: change it to w
- replace X_i in rules by $a^{\ell_i} X_i b^{r_i}$
- analysis: more tricky but works
- $\mathcal{O}(g)$

In total

- $\mathcal{O}(g)$ per phase
- $\mathcal{O}(\log n)$ phases
- $\mathcal{O}(g \log n)$ credit in total (= size of created grammar)
- can be improved to $\mathcal{O}(g \log(n/g))$

Acknowledgments

M. Lohrey

Suggesting the analysis.

Acknowledgments

M. Lohrey

Suggesting the analysis.

P. Gawrychowski

- introducing to the topic
- literature
  - K. Mehlhorn, R. Sundar and Ch. Uhrig, *Maintaining Dynamic Sequences under Equality Tests in Polylogarithmic Time*, ‘97
  - H. Sakamoto, *A fully linear-time approximation algorithm for grammar-based compression*, ’05
  - M. Lohrey and Ch. Mathissen, *Compressed Membership in Automata with Compressed Labels*, ’11

Open problems, related research

Open problems

- better approximation
- simpler computational model (no RadixSort)
- addition chains ($\mathcal{O}(\frac{\log n}{\log \log n})$ approximation known)

Open problems, related research

Open problems

- better approximation
- simpler computational model (no RadixSort)
- addition chains ($\mathcal{O}(\frac{\log n}{\log \log n})$ approximation known)

Other applications: recompression

- compressed membership
- fully compressed pattern matching
- word equations