Smallest tree grammar by recompression

Artur Jeż \textsuperscript{1}  Markus Lohrey\textsuperscript{2}

Max Planck Institute für Informatik

University of Siegen

07.03.2014
Compression and grammars

Compression

- Increasingly popular
- many approaches
## Compression and grammars

### Compression
- Increasingly popular
- many approaches

### Grammars based compression
- CFG defining unique word
- Straight Line Programs (SLP)
- easy to work on
- natural in many applications
- Block-based compression translates to SLPs.
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 \( L(G_w) = w \).

- decision problem: NP-hard
- lower bound for approximation ratio

Best approximation ratio
\( \mathcal{O}(\log(n/g)) \), where \( g \) is the size of the optimal grammar.
Tree grammars

Trees

What about grammars for (labelled) trees?

Definition (labelled trees = terms)

\[ \Sigma = \bigcup_{i \geq 0} \Sigma^i \]

\[ \text{rank} : \Sigma \rightarrow \mathbb{N}, \text{rank}(\Sigma^i) = \{i\} \]

rooted trees, nodes labelled with elements of \( \Sigma \)

node \( a \) has rank \( \text{rank}(a) \) children 

Grammar: different generalisations.
Tree grammars

Trees

What about grammars for (labelled) trees?

Definition (labelled trees = terms)

- (ranked) alphabet $\Sigma = \bigcup_{i \geq 0} \Sigma_i$
- $\text{rank} : \Sigma \rightarrow \mathbb{N}$, $\text{rank}(\Sigma_i) = \{i\}$
- rooted trees, nodes labelled with elements of $\Sigma$
- node $a$ has $\text{rank}(a)$ children (ordered)
Tree grammars

Trees

What about grammars for (labelled) trees?

Definition (labelled trees = terms)

- (ranked) alphabet $\Sigma = \bigcup_{i \geq 0} \Sigma_i$
- rank : $\Sigma \mapsto \mathbb{N}$, $\text{rank}(\Sigma_i) = \{i\}$
- rooted trees, nodes labelled with elements of $\Sigma$
- node $a$ has rank$(a)$ children (ordered)

Grammar: different generalisations.
SLPs for strings

Definition (SLP: Straight Line Programme)

CFG with
- ordered nonterminals $X_1, X_2, \ldots$
- Chomsky normal form
- one rule for nonterminal
- for $X_i \rightarrow X_jX_k$ we have $j, k < i$
From string to trees

Simplest

- ordered nonterminals $X_1, X_2, \ldots$
  each generates a tree
- rules $X_i \rightarrow f(X_j, \ldots, X_k)$ we have $j, \ldots, k < i$
From string to trees

**Simplest**
- ordered nonterminals $X_1, X_2, \ldots$
each generates a tree
- rules $X_i \rightarrow f(X_j, \ldots, X_k)$ we have $j, \ldots, k < i$

**DAGs**
- those are exactly DAGs
- smallest one can be found
From string to trees

**Simplest**
- ordered nonterminals $X_1, X_2, \ldots$
  each generates a tree
- rules $X_i \rightarrow f(X_j, \ldots, X_k)$ we have $j, \ldots, k < i$

**DAGs**
- those are exactly DAGs
- smallest one can be found

Not a good candidate
SLCF grammar

SLP rewrites nonterminals keeping left and right ‘context’.

SLCF grammar
SLCF grammar

SLP rewrites nonterminals keeping left and right ‘context’.

- ordered ranked nonterminals $X_1, X_2, \ldots$
- rules $X_i(y_1, y_2, \ldots, y_m) \rightarrow t$, where
  - $m$ is the arity of $X_i$
  - $t$ contains a single leaf $y_1, \ldots, y_m$
  - may contain $X_1, \ldots, X_{i-1}$

$$A(y_1, y_2, y_3) \rightarrow t$$

![Diagram](https://via.placeholder.com/150)

07.03.2014
SLCF grammar

SLP rewrites nonterminals keeping left and right ‘context’.

SLCF grammar

- ordered ranked nonterminals $X_1, X_2, \ldots$
- rules $X_i(y_1, y_2, \ldots, y_m) \rightarrow t$, where
  - $m$ is the arity of $X_i$
  - $t$ contains a single leaf $y_1, \ldots, y_m$
  - may contain $X_1, \ldots, X_{i-1}$
- if $y_i$ may be multiplied: turns very difficult.

![Diagram of SLCF grammar](image)
Properties and intuition

Grammar for text: 1 parameter (text to the right)

\[ A \rightarrow w \iff A(y) \rightarrow w(y) \]
Properties and intuition

Grammar for text: 1 parameter (text to the right)
\[ A \to w \iff A(y) \to w(y) \]

Compression ratio
- at most exponential
- exponential for some cases
Properties and intuition

Grammar for text: 1 parameter (text to the right)
\[ A \rightarrow w \iff A(y) \rightarrow w(y) \]

Compression ratio
- at most exponential
- exponential for some cases

Lemma (Lohrey, Maneth, Schauss-Schmidt)

*Without loss of generality each nonterminal has 0 or 1 parameter. Size increases \( O(r) \) times.*

Proof.
Rightmost derivation.
Smallest tree grammar

First problem: smallest grammar

Given a labelled tree $T$ return smallest SLCF generating it.
Smallest tree grammar

First problem: smallest grammar

Given a labelled tree $T$ return smallest SLCF generating it.

Simply generalise the algorithm for strings.
Smallest tree grammar

First problem: smallest grammar

Given a labelled tree $T$ return smallest SLCF generating it.

Simply generalise the algorithm for strings.

- All those algorithms use LZ77. Does not generalise to trees.
Smallest tree grammar

First problem: smallest grammar

Given a labelled tree $T$ return smallest SLCF generating it.

Simply generalise the algorithm for strings.

- All those algorithms use LZ77. Does not generalise to trees.
- Recent not LZ77 based algorithm.

Theorem (NEW!)

The smallest tree grammar can be $O(r \log N)$ approximated, where $r$ is the maximal rank.
Smallest tree grammar

First problem: smallest grammar

Given a labelled tree $T$ return smallest SLCF generating it.

Simply generalise the algorithm for strings.

- All those algorithms use LZ77. Does not generalise to trees.
- Recent not LZ77 based algorithm.

Theorem (NEW!)

The smallest tree grammar can be $O(r \log N)$ approximated, where $r$ is the maximal rank.

- based on local compression rules
- analysis modifies the optimal grammar
Leaf compression

- ‘Absorb’ each leaf by its father (and change the labels).
- Replace $f(c_1, t_2, t_3, \ldots, c_2, \ldots, t_k)$ with $f'(t_2, t_3, \ldots, t_k)$
Leaf compression

- ‘Absorb’ each leaf by its father (and change the labels).
- Replace $f(c_1, t_2, t_3, \ldots, c_2, \ldots, t_k)$ with $f'(t_2, t_3, \ldots, t_k)$
- Rule $f'(y_2, y_3, \ldots, y_k) \rightarrow f(c_1, y_2, y_3, \ldots, c_2, \ldots, y_k)$
Leaf compression

- ‘Absorb’ each leaf by its father (and change the labels).
- Replace $f(c_1, t_2, t_3, \ldots, c_2, \ldots, t_k)$ with $f'(t_2, t_3, \ldots, t_k)$
- rule $f'(y_2, y_3, \ldots, y_k) \rightarrow f(c_1, y_2, y_3, \ldots, c_2, \ldots, y_k)$
Leaf compression

- ‘Absorb’ each leaf by its father (and change the labels).
- Replace \( f(c_1, t_2, t_3, \ldots, c_2, \ldots, t_k) \) with \( f'(t_2, t_3, \ldots, t_k) \)
- Rule \( f'(y_2, y_3, \ldots, y_k) \rightarrow f(c_1, y_2, y_3, \ldots, c_2, \ldots, y_k) \)

- Half of nodes are leaves
- \( \mathcal{O}(\log n) \) rounds
Chains compression

- Long chains are not really affected

\[ a \in \Sigma^1 \cap \Sigma^1' = \emptyset \]
Chains compression

- Long chains are not really affected
- those are almost strings
- for strings we know what to do

2-chain compression: replace $ab$ with $a'$

$a_k$-chain compression: replace $a_k$

2-chain compression: $ab \in \Sigma_1 \Sigma_1'$, where $\Sigma_1 \cap \Sigma_1' = \emptyset$

Lemma 1/4 of the nodes are compressed.
Chains compression

- Long chains are not really affected
- those are almost strings
- for strings we know what to do
  - 2-chain compression: replace $ab$ with $a'$
  - $a$-chain compression: replace $a^k$ with $a_k$
Chains compression

- Long chains are not really affected
- those are almost strings
- for strings we know what to do
  - 2-chain compression: replace $ab$ with $a'$
  - $a$-chain compression: replace $a^k$ with $a_k$
- 2-chain compression: $ab \in \Sigma_1 \Xi'$, where $\Sigma_1 \cap \Sigma'_1 = \emptyset$
Chains compression

- Long chains are not really affected
- those are almost strings
- for strings we know what to do
  - 2-chain compression: replace $ab$ with $a'$
  - $a$-chain compression: replace $a^k$ with $a_k$
- 2-chain compression: $ab \in \Sigma \Sigma'$, where $\Sigma \cap \Sigma' = \emptyset$

**Lemma**

1/4 of the nodes are compressed.
Algorithm

1: while $|T| > 1$ do

2: $L \leftarrow$ list of unary letters in $T$

3: for each $a \in L$ do

4: $\Delta a$-chain compression

5: compress maximal chains of $a$

6: $P \leftarrow$ list of 2-chains

7: find partition of $\Sigma$ into $\Sigma^l$ and $\Sigma^r$

8: for $ab \in P \cap \Sigma^l \Sigma^r$ do

9: $\Delta$ these 2-chains do not overlap

10: compress 2-chain $ab$

11: $\Delta 2$-chains compression

12: $L_0 \leftarrow$ list of constants,

13: $L \geq 1 \leftarrow$ list of other letters in $T$

14: for $f \in L \geq 1$ and $1 \leq \ell \leq \text{rank}(f)$ and $a \in L_0$ do

15: perform all leaf compressions for $f a$
Algorithm

1: while $|T| > 1$ do
2: \hfill $L \leftarrow$ list of unary letters in $T$
3: \hspace{1em} for each $a \in L$ do $\triangleright a$-chain compression
4: \hspace{2em} compress maximal chains of $a$

5: $P \leftarrow$ list of 2-chains
6: find partition of $\Sigma$ into $\Sigma_\ell$ and $\Sigma_r$
7: for $ab \in P \cap \Sigma_\ell \Sigma_r$ do $\triangleright$ These 2-chains do not overlap
8: \hspace{1em} compress 2-chain $ab$ $\triangleright$ 2-chains compression
9: $L_0 \leftarrow$ list of constants, $L \geq 1 \leftarrow$ list of other letters in $T$
10: for $f \in L \geq 1$ and $1 \leq \ell \leq \text{rank}(f)$ and $a \in L_0$ do
11: \hspace{1em} perform all leaf compressions for $f \ a$
1: while $|T| > 1$ do
2: \hspace{1cm} $L \leftarrow$ list of unary letters in $T$
3: \hspace{1cm} for each $a \in L$ do \hspace{1cm} $\triangleright$ a-chain compression
4: \hspace{1cm} \hspace{1cm} compress maximal chains of $a$
5: \hspace{1cm} $P \leftarrow$ list of 2-chains
6: \hspace{1cm} find partition of $\Sigma$ into $\Sigma_\ell$ and $\Sigma_r$
7: \hspace{1cm} for $ab \in P \cap \Sigma_\ell \Sigma_r$ do \hspace{1cm} $\triangleright$ These 2-chains do not overlap
8: \hspace{1cm} \hspace{1cm} compress 2-chain $ab$ \hspace{1cm} $\triangleright$ 2-chains compression
Algorithm

1: while $|T| > 1$ do
2:     $L \leftarrow$ list of unary letters in $T$
3:     for each $a \in L$ do \hspace*{1cm} $\triangleright$ a-chain compression
4:         compress maximal chains of $a$
5:     $P \leftarrow$ list of 2-chains
6:     find partition of $\Sigma$ into $\Sigma_\ell$ and $\Sigma_r$
7:     for $ab \in P \cap \Sigma_\ell \Sigma_r$ do \hspace*{1cm} $\triangleright$ These 2-chains do not overlap
8:         compress 2-chain $ab$ \hspace*{1cm} $\triangleright$ 2-chains compression
9:     $L_0 \leftarrow$ list of constants, $L_{\geq 1} \leftarrow$ list of other letters in $T$
10:    for $f \in L_{\geq 1}$ and $1 \leq \ell \leq \text{rank}(f)$ and $a \in L_0$ do
11:       perform all leaf compressions for $f \; a$
Lemma

*In one phase the size drops by a constant factor.*
Time and size analysis

Lemma

*In one phase the size drops by a constant factor.*

- **no chain** we remove all leaves, size halves
- **single chain** a string, size drops by a constant factor
- **general** some mix of above
# Time and size analysis

## Lemma

*In one phase the size drops by a constant factor.*

| **no chain** | we remove all leaves, size halves |
| **single chain** | a string, size drops by a constant factor |
| **general** | some mix of above |

## Time

- enough if one phase takes linear time.
- compressions: grouping done by sorting (RadixSort)
Size analysis

- Modifications of the smallest SLCF.
- This is known for the string case.
Size analysis

- Modifications of the smallest SLCF.
- This is known for the string case.

Mental experiment

- We take the smallest SLCF
- We perform the compression step on it
  - it always generates the current tree
- Some changes of the SLCF are needed
- The number of nonterminals depends on the SLCF, not tree.
Compression on the grammar

Perform the compression step on the grammar. Eg. $a(b(\cdot)) \rightarrow c(\cdot)$
Compression on the grammar

Perform the compression step on the grammar.
Eg. $a(b(\cdot)) \rightarrow c(\cdot)$

Bounding the cost

- each letter has credit
- during compression credit is released
  - $ab$ has 2 credit, $c$ only 1
- it pays for the rule for the new letter $c(y) \rightarrow a(b(y))$
Compression on the grammar

Perform the compression step on the grammar. Eg. $a(b(·)) \rightarrow c(·)$

Bounding the cost

- each letter has credit
- during compression credit is released
  - $ab$ has 2 credit, $c$ only 1
- it pays for the rule for the new letter $c(y) \rightarrow a(b(y))$

Ensure that this is OK

- $X(y) \rightarrow aa(y)$, $Y \rightarrow X(b)$: $ab$ not there
Rules modification (recompression)

Modification of the rules

- $X(y) \rightarrow a\ a(y)$, $Y \rightarrow X(b)$ to
- $X \rightarrow a(y)$, $Y \rightarrow X(ab)$: ab is OK.

This increases the credit.
Rules modification (recompression)

Modification of the rules

- $X(y) \rightarrow aa(y)$, $Y \rightarrow X(b)$ to
- $X \rightarrow a(y)$, $Y \rightarrow X(ab)$: ab is OK.

This increases the credit.

- Total cost: the issued credit
- $O(rg)$ credit per phase. Essentially: $O(r)$ per nonterminal.
- $O(rg \log N)$ in total
Similar results and open problems

Other applications

Applies also to context unification.
Similar results and open problems

### Other applications

Applies also to **context unification**.

### Open problems

- **Lower bound**
  - only constant lower bound for approximation ratio
  - already for (very simple) strings

- **What is the approximation bound**
  - strings
  - trees
Similar results and open problems

Other applications
Applies also to context unification.

Open problems
- Lower bound
  - only constant lower bound for approximation ratio
  - already for (very simple) strings
- What is the approximation bound
  - strings
  - trees

More general grammar — hardness?