Smallest tree grammar by recompression

Artur Jeż [1] **Markus Lohrey**[2]

Max Planck Institute fur 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?

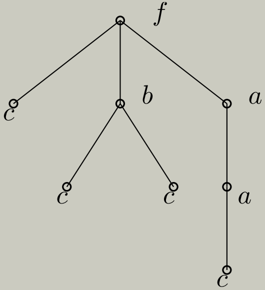
Tree grammars

Trees

What about grammars for (labelled) trees?

Definition (labelled trees = terms)

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



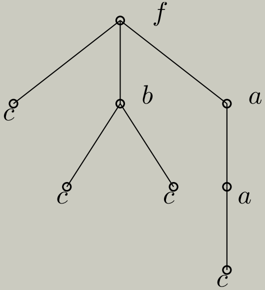
Tree grammars

Trees

What about grammars for (labelled) trees?

Definition (labelled trees = terms)

- (ranked) alphabet $\Sigma = \bigcup_{i \geq 0} \Sigma_i$
- $\text{rank} : \Sigma \mapsto \mathbb{N}$, $\text{rank}(\Sigma_i) = \{i\}$
- rooted trees, nodes labelled with elements of Σ
- 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 $X_i \to X_j X_k$ 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 $X_1, X_2, \dots$
  each generates a tree
- rules $X_i \to f(X_j, \dots, X_k)$ we have $j, \dots, k < i$

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 $X_i \to f(X_j, \ldots, X_k)$ 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’.

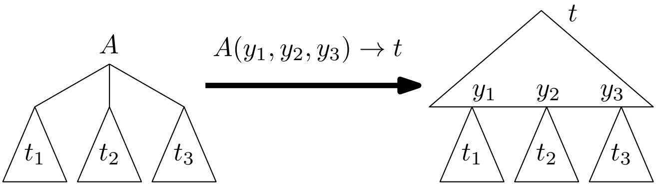
SLCF grammar

SLCF grammar

SLP rewrites nonterminals keeping left and right ‘context’.

SLCF grammar

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

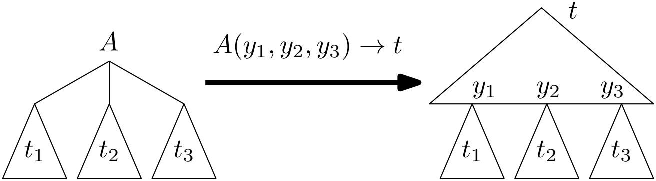


SLCF grammar

SLP rewrites nonterminals keeping left and right ‘context’.

SLCF grammar

- ordered ranked nonterminals $X_1, X_2, \ldots$
- rules $X_i(y_1, y_2, \ldots, y_m) \to t$, 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.



Properties and intuition

Grammar for text: 1 parameter (text to the right)
$A \rightarrow w \Leftrightarrow A(y) \rightarrow w(y)$

Properties and intuition

Grammar for text: 1 parameter (text to the right)
$A \rightarrow w \Leftrightarrow A(y) \rightarrow w(y)$

Compression ratio

- at most exponential
- exponential for some cases

Properties and intuition

Grammar for text: 1 parameter (text to the right)
$A \rightarrow w \Leftrightarrow A(y) \rightarrow w(y)$

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 $\mathcal{O}(r)$ *times.*

Proof.

Rightmost derivation. $\square$

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 $\mathcal{O}(r \log N)$ 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 $\mathcal{O}(r \log N)$ approximated, where r is the maximal rank.

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

Leaf compression

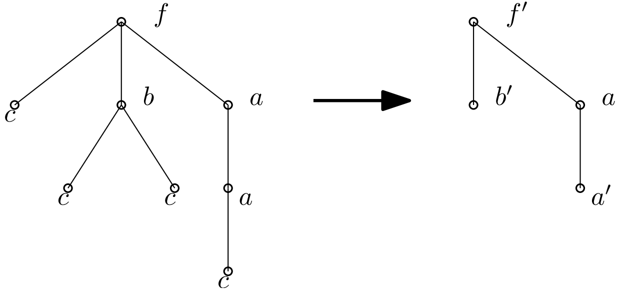
- 'Absorb' each leaf by its father (and change the labels).
- Replace $f(c_1, t_2, t_3, \ldots, c_2, \ldots, t_k)$ with $f'(t_2, t_3, \ldots, t_k)$

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 $f'(y_2, y_3, \ldots, y_k) \to f(c_1, y_2, y_3, \ldots, c_2, \ldots, y_k)$

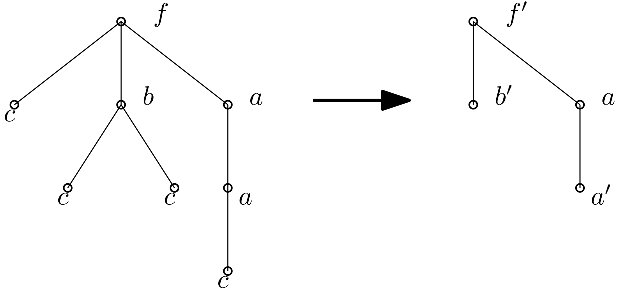
Leaf compression

- 'Absorb' each leaf by its father (and change the labels).
- Replace $f(c_1, t_2, t_3, \dots, c_2, \dots, t_k)$ with $f'(t_2, t_3, \dots, t_k)$
- rule $f'(y_2, y_3, \dots, y_k) \to f(c_1, y_2, y_3, \dots, c_2, \dots, y_k)$



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 $f'(y_2, y_3, \ldots, y_k) \to f(c_1, y_2, y_3, \ldots, c_2, \ldots, y_k)$



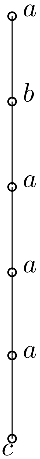
- half of nodes are leaves
- $\mathcal{O}(\log n)$ rounds

Chains compression

- Long chains are not really affected

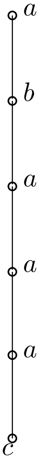
Chains compression

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



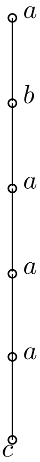
Chains compression

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

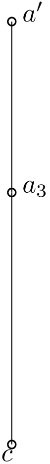


Chains compression

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

Lemma

1/4 of the nodes are compressed.



Algorithm

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

Algorithm

while $|T| > 1$ **do**
  $L \leftarrow$ list of unary letters in T
  for each $a \in L$ **do** $\triangleright$ *a*-chain compression
    compress maximal chains of a

Algorithm

while $|T| > 1$ **do**
  $L \leftarrow$ list of unary letters in T
  for each $a \in L$ **do** $\triangleright$ a-chain compression
    compress maximal chains of a
  $P \leftarrow$ list of 2-chains
  find partition of Σ into Σ_ℓ and Σ_r
  for $ab \in P \cap \Sigma_\ell \Sigma_r$ **do** $\triangleright$ These 2-chains do not overlap
    compress 2-chain ab $\triangleright$ 2-chains compression

Algorithm

while $|T| > 1$ **do**
  $L \leftarrow$ list of unary letters in T
  for each $a \in L$ **do** ▷ a-chain compression
    compress maximal chains of a
  $P \leftarrow$ list of 2-chains
  find partition of Σ into Σ_ℓ and Σ_r
  for $ab \in P \cap \Sigma_\ell \Sigma_r$ **do** ▷ These 2-chains do not overlap
    compress 2-chain ab ▷ 2-chains compression
  $L_0 \leftarrow$ list of constants, $L_{\geq 1} \leftarrow$ list of other letters in T
  for $f \in L_{\geq 1}$ and $1 \leq \ell \leq \text{rank}(f)$ and $a \in L_0$ **do**
    perform all leaf compressions for $f\ a$

Time and size analysis

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. $a(b(\cdot)) \to c(\cdot)$

Compression on the grammar

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

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 $c(y) \to a(b(y))$

Compression on the grammar

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

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 $c(y) \to a(b(y))$

Ensure that this is OK

- $X(y) \to aa(y)$, $Y \to X(b)$: ab not there

Rules modification (recompression)

Modification of the rules

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

This increases the credit.

Rules modification (recompression)

Modification of the rules

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

This increases the credit.

- Total cost: the issued credit
- $\mathcal{O}(rg)$ credit per phase. Essentially: $\mathcal{O}(r)$ per nonterminal.
- $\mathcal{O}(rg \log N)$ 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?