Clique-Based Lower Bounds for Parsing Tree-Adjoining Grammars

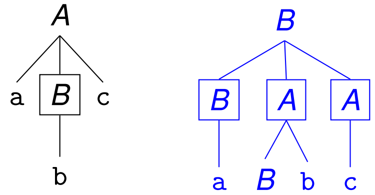
Karl Bringmann and **Philip Wellnitz**

Max Planck Institute for Informatics,
Saarland Informatics Campus (SIC),
Saarbrücken, Germany

May 8, 2019

Tree-Adjoining Grammars

- Initial and auxiliary trees, nodes marked for adjunction

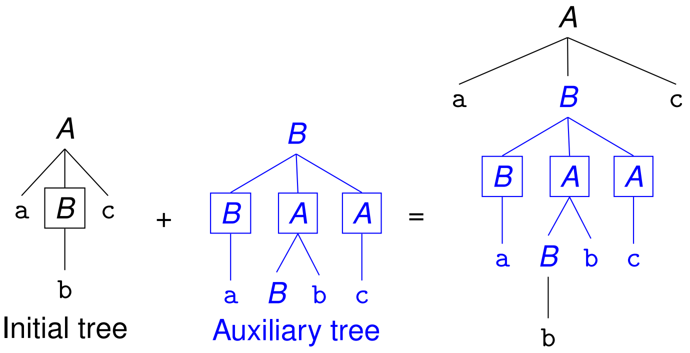


Initial tree

Auxiliary tree

Tree-Adjoining Grammars

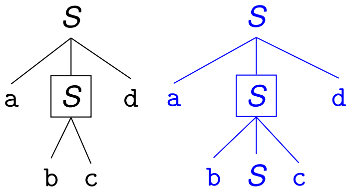
- Adjoining trees



Tree-Adjoining Grammars

Examples

- Every CFG
- $\{a^n b^n c^n d^n \mid n \in \mathbb{N}\}$



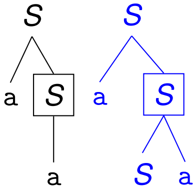
- (Large parts of) English (XTAG [DEH+94])

Tree-Adjoining Grammars

Examples

- Every CFG
- $\{a^n b^n c^n d^n \mid n \in \mathbb{N}\}$
- (Large parts of) English (XTAG [DEH+94])

$\{aa \mid a \in \Sigma\}$



Tree-Adjoining Grammars

- Check for $s \in T^*$ if $s \in \mathcal{L}(\Gamma)$, i.e. *parse s*
- $O(|s|^6)$ algorithm using dynamic programming [VSJ85, SJ88]
- $O(|s|^{2\omega})$ using matrix multiplication, $\omega < 2.373$ [RY98]

- Faster algorithms?
  $\rightsquigarrow$ Improbable, for $|\Gamma| = \Theta(n^{12})$ [Sat94]
  $\rightsquigarrow$ *Now, even for* $|\Gamma| = \Theta(1)$

Tree-Adjoining Grammars

- Check for $s \in T^*$ if $s \in \mathcal{L}(\Gamma)$, i.e. *parse s*
- $O(|s|^6)$ algorithm using dynamic programming [VSJ85, SJ88]
- $O(|s|^{2\omega})$ using matrix multiplication, $\omega < 2.373$ [RY98]

- Faster algorithms?
  $\rightsquigarrow$ Improbable, for $|\Gamma| = \Theta(n^{12})$ [Sat94]
  $\rightsquigarrow$ *Now, even for* $|\Gamma| = \Theta(1)$

Tree-Adjoining Grammars

- Check for $s \in T^*$ if $s \in \mathcal{L}(\Gamma)$, i.e. *parse s*
- $O(|s|^6)$ algorithm using dynamic programming [VSJ85, SJ88]
- $O(|s|^{2\omega})$ using matrix multiplication, $\omega < 2.373$ [RY98]

- Faster algorithms?
  $\rightsquigarrow$ Improbable, for $|\Gamma| = \Theta(n^{12})$ [Sat94]
  $\rightsquigarrow$ *Now, even for* $|\Gamma| = \Theta(1)$

Tree-Adjoining Grammars

- Check for $s \in T^*$ if $s \in \mathcal{L}(\Gamma)$, i.e. *parse s*
- $O(|s|^6)$ algorithm using dynamic programming [VSJ85, SJ88]
- $O(|s|^{2\omega})$ using matrix multiplication, $\omega < 2.373$ [RY98]

- Faster algorithms?
  $\rightsquigarrow$ Improbable, for $|\Gamma| = \Theta(n^{12})$ [Sat94]
  $\rightsquigarrow$ *Now, even for* $|\Gamma| = \Theta(1)$

Lower Bounds

- Hard to show lower bounds for natural problems

$\rightsquigarrow$ Use *reductions* to relate problems

Problem P
Instance I

size n

reduction
$\longrightarrow$
$r(n)$ time

Problem Q
Instance J

$$(x_1 \vee \overline{x_3} \vee x_5) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

size $s(n)$

I is "yes" instance $\iff$ J is "yes" instance

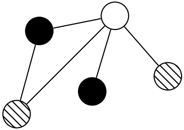
$r(n) + t(s(n))$ algorithm $\longleftarrow$ $t(n)$ algorithm

Lower Bounds

- Hard to show lower bounds for natural problems
  $\rightsquigarrow$ Use *reductions* to relate problems

Problem P

Instance I



size n

reduction

$\longrightarrow$

$r(n)$ time

Problem Q

Instance J

$$(x_1 \vee \overline{x_3} \vee x_5) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

size $s(n)$

I is "yes" instance $\iff$ J is "yes" instance

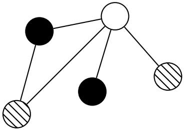
$r(n) + t(s(n))$ algorithm $\longleftarrow$ $t(n)$ algorithm

Lower Bounds

- Hard to show lower bounds for natural problems
  $\rightsquigarrow$ Use *reductions* to relate problems

Problem P

Instance I



size n

reduction $\longrightarrow$ $r(n)$ time

Problem Q

Instance J

$$(x_1 \vee \overline{x_3} \vee x_5) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

size $s(n)$

I is "yes" instance $\iff$ J is "yes" instance

No $r(n) + t(s(n))$ algorithm $\longrightarrow$ **No $t(n)$ algorithm**

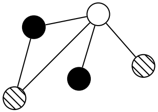
Lower Bounds

- Hard to show lower bounds for natural problems

$\rightsquigarrow$ Use *reductions* to relate problems

Problem P

Instance I



size n

reduction $\longrightarrow$ $r(n)$ time

Problem Q

Instance J

$$(x_1 \vee \overline{x_3} \vee x_5) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

size $s(n)$

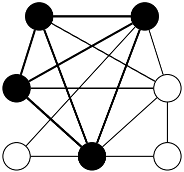
I is "yes" instance $\iff$ J is "yes" instance

No $r(n) + t(s(n))$ **algorithm** $\longrightarrow$ **No** $t(n)$ **algorithm**

$\rightsquigarrow$ Need *hard* problems

k-Clique

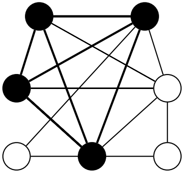
- Given a graph $G = (V, E)$ and $k \in \mathbb{N}$, does G contain a clique of size k?



- Naïve $O(n^k)$ algorithm
- $O(n^{\frac{\omega k}{3}})$ for $3 \mid k$, using matrix multiplication, $\omega < 2.373$ [NP85]
- **Let us believe that these algorithms are optimal.**

k-Clique

- Given a graph $G = (V, E)$ and $k \in \mathbb{N}$, does G contain a clique of size k?

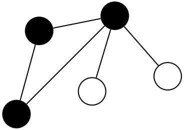


- Naïve $O(n^k)$ algorithm
- $O(n^{\frac{\omega k}{3}})$ for $3 \mid k$, using matrix multiplication, $\omega < 2.373$ [NP85]
- **Let us believe that these algorithms are optimal.**

Hardness for Context-Free Grammars

3k-Clique

$G = (V, E)$, k



size $n = |V|$

reduction → n^{k+1} time

Parsing CFG

CFG Γ, string s

$\Gamma = (T, NT, P, S)$,

graphs w/ 3k-Cliques

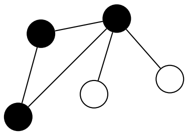
s, encoding of G

size $s(n) = |s| = n^{k+1}$

G contains 3k-Clique $\iff$ $s \in \mathcal{L}(\Gamma)$

No $n^{k+1} + t(n^{k+1})$ algorithm $\longrightarrow$ **No $t(n)$ algorithm**

Hardness for Context-Free Grammars

$3k$-Clique		**Parsing CFG**
$G = (V, E)$, k		CFG Γ, string s
	*reduction* $\longrightarrow$ n^{k+1} time	$\Gamma = (T, NT, P, S)$, graphs w/ $3k$-Cliques; s, encoding of G
size $n = \|V\|$		size $s(n) = \|s\| = n^{k+1}$

G contains $3k$-Clique $\iff$ $s \in \mathcal{L}(\Gamma)$

No $n^{3k(1-\varepsilon)}$ algorithm $\longrightarrow$ **No $n^{3-\varepsilon'}$ algorithm**

No $n^{\frac{\omega k}{3}(1-\varepsilon)}$ algorithm $\longrightarrow$ No $n^{\omega-\varepsilon'}$ algorithm

Hardness for Context-Free Grammars

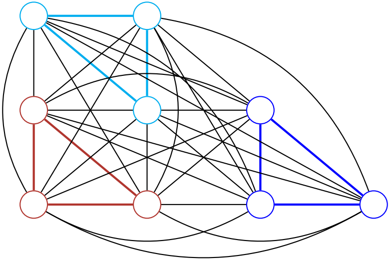
- String s: list all k-cliques of G in a special way

  $\rightsquigarrow$ Γ: all graphs where 3 k-cliques form a triangle

Hardness for Context-Free Grammars

- String s: list all k-cliques of G in a special way

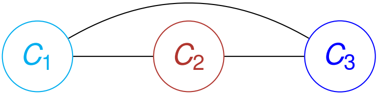
$\rightsquigarrow$ Γ: all graphs where 3 k-cliques form a triangle



Hardness for Context-Free Grammars

- String s: list all k-cliques of G in a special way

$\rightsquigarrow$ Γ: all graphs where 3 k-cliques form a triangle



$$(+|+)_2\ (+|+)_3\ (+|+)_1 \cdots (-|+)_2\ (-|+)_1\ (-|+)_3 \cdots (-|-)_1\ (-|-)_3\ (-|-)_2$$

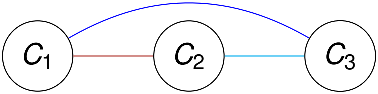
$+$: List clique's nodes

$-$: List for every node all neighbors

Hardness for Context-Free Grammars

- String s: list all k-cliques of G in a special way

$\rightsquigarrow$ Γ: all graphs where 3 k-cliques form a triangle



$$\cdots \quad (+|+)_1 \quad \cdots \quad (-|+)_2 \quad \cdots \quad (-|-)_3 \quad \cdots$$

$+$: List clique's nodes $\qquad$ $-$: List for every node all neighbors

Matching $+$ and $-$ form a $2k$-clique.

Γ will look like:

$$S \rightarrow^* \cdots (+|S_{\alpha\beta}|S_{\beta\gamma}|-) \cdots$$
$$S_{\alpha\beta} \rightarrow^* +) \cdots (-$$
$$S_{\beta\gamma} \rightarrow^* +) \cdots (-$$

Hardness for Context-Free Grammars

Defining suitable gadgets:

- Node gadget $\mathrm{NG}(v) := \$\,\mathrm{binary}(v)\,\$$
- List gadget $\mathrm{LG}(v) := \bigcirc_{u \in N(v)} \mathrm{NG}(u)$
- Clique node gadget $(+)$ $\mathrm{CNG}(C) := \bigcirc_{v \in C} (\#\mathrm{NG}(v)\#)^k$
- Clique list gadget $\mathrm{CLG}(C) := \left(\bigcirc_{v \in C} \#\mathrm{LG}(v)\# \right)^k$

$\rightsquigarrow$ Use $\mathrm{CLG}(C)^R$ for $-$ gadgets

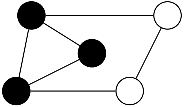
- $\mathrm{CNG}(C_1) \subseteq \mathrm{CLG}(C_2) \Rightarrow C_1 \cup C_2$ is a $2k$-clique

$\rightsquigarrow$ CFGs can generate aa^R

Hardness for Tree-Adjoining Grammars

$6k$-Clique

$G = (V, E)$, k



size $n = |V|$

reduction

$\longrightarrow$

n^{k+1} *time*

Parsing TAG

TAG Γ, string s

$\Gamma = (I, A)$,
graphs w/ $6k$-cliques
s, encoding of G

size $|s| = n^{k+1}$

G contains $6k$-clique $\iff$ $s \in \mathcal{L}(\Gamma)$

No $n^{6k(1-\varepsilon')}$ algorithm
No $n^{2\omega k(1-\varepsilon')}$ algorithm
$\longrightarrow$
No $|s|^{6-\varepsilon}$ algorithm
No $|s|^{2\omega-\varepsilon}$ algorithm

Hardness for Tree-Adjoining Grammars

- Want to "solve" $6k$-clique using TAG parser
  $\rightsquigarrow$ Need string and a grammar
- String: list all k-cliques in graph G in a special way
  Grammar: all graphs where 6 k-cliques form a 6-clique
- **Here:** Will show how to generate $4k$-cliques

Hardness for Tree-Adjoining Grammars

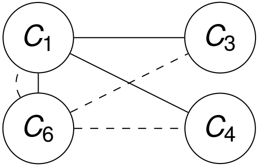
- Want to “solve” $6k$-clique using TAG parser
  $\rightsquigarrow$ Need string and a grammar
- String: list all k-cliques in graph G in a special way
  Grammar: all graphs where 6 k-cliques form a 6-clique
- **Here:** Will show how to generate $4k$-cliques

Hardness for Tree-Adjoining Grammars

- Can generate (almost) $4k$-cliques $\rightsquigarrow$ $\mathrm{P}(\cdot,\cdot,\cdot,\cdot)$
- $6k$-cliques decompose into 3 of these

  $\rightsquigarrow$ Need to use $\mathrm{P}(\cdot,\cdot,\cdot,\cdot)$ 3 times

  $\rightsquigarrow$ Similar to CFG hardness

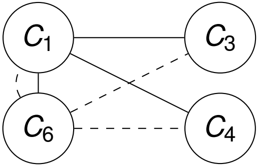


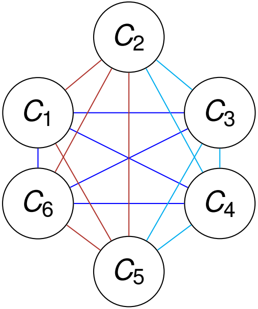
Hardness for Tree-Adjoining Grammars

- Can generate (almost) $4k$-cliques $\rightsquigarrow$ $\mathrm{P}(\cdot,\cdot,\cdot,\cdot)$
- $6k$-cliques decompose into 3 of these

$\rightsquigarrow$ Need to use $\mathrm{P}(\cdot,\cdot,\cdot,\cdot)$ 3 times

$\rightsquigarrow$ Similar to CFG hardness



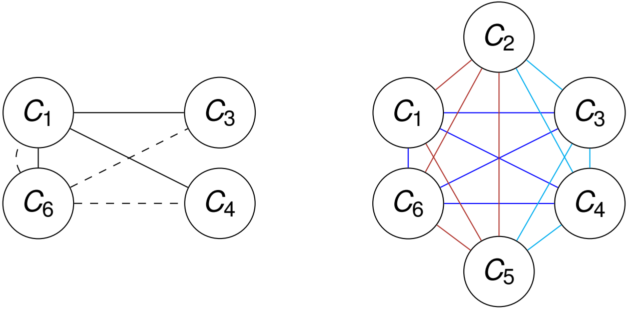


Hardness for Tree-Adjoining Grammars

- Can generate (almost) $4k$-cliques $\rightsquigarrow$ $\mathrm{P}(\cdot,\cdot,\cdot,\cdot)$
- $6k$-cliques decompose into 3 of these

$\rightsquigarrow$ Need to use $\mathrm{P}(\cdot,\cdot,\cdot,\cdot)$ 3 times

$\rightsquigarrow$ Similar to CFG hardness



Hardness for Tree-Adjoining Grammars

- String: list all k-cliques in graph G in a special way
  Grammar: all graphs where 6 k-cliques form a 6-clique
- $\mathrm{CNG}(C_k)$ list vertices of k-clique C_k
  $\mathrm{CLG}(C_k)$ list neighbors of vertices of k-clique C_k
- $\mathrm{CNG}(C_k) \subset \mathrm{CLG}(C'_k)$ iff $C_k \cup C'_k$ is a $2k$-clique

Hardness for Tree-Adjoining Grammars

- String: list all k-cliques in graph G in a special way
  Grammar: all graphs where 6 k-cliques form a 6-clique
- $\mathrm{CNG}(C_k)$ list vertices of k-clique C_k
  $\mathrm{CLG}(C_k)$ list neighbors of vertices of k-clique C_k
- $\mathrm{CNG}(C_k) \subset \mathrm{CLG}(C'_k)$ iff $C_k \cup C'_k$ is a $2k$-clique

Hardness for Tree-Adjoining Grammars

- $\mathrm{CNG}(C_k)$ list vertices of k-clique C_k
  $\mathrm{CLG}(C_k)$ list neighbors of vertices of k-clique C_k
- $\mathrm{CNG}(C_k) \subset \mathrm{CLG}(C'_k)$ iff $C_k \cup C'_k$ is a $2k$-clique
- String (to detect $4k$-cliques):

$$\begin{aligned}\mathrm{GG_k}(G) := & \bigcirc_{C \in \mathcal{C}_k} |\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R| \\ & \circ \bigcirc_{C \in \mathcal{C}_k} |\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R| \\ & \circ \bigcirc_{C \in \mathcal{C}_k} |\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R| \\ & \circ \bigcirc_{C \in \mathcal{C}_k} |\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R|\end{aligned}$$

Hardness for Tree-Adjoining Grammars

- $\mathrm{CNG}(C_k) \subset \mathrm{CLG}(C'_k)$ iff $C_k \cup C'_k$ is a $2k$-clique
- String (to detect $4k$-cliques):

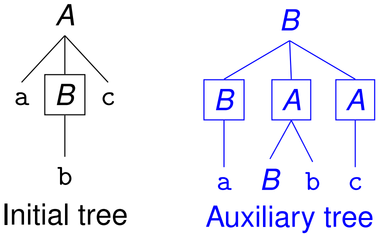
$$\begin{aligned}
\mathrm{GG_k}(G) := & \bigcirc_{C \in \mathcal{C}_k} |\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R| \\
\circ & \bigcirc_{C \in \mathcal{C}_k} |\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R| \\
\circ & \bigcirc_{C \in \mathcal{C}_k} |\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R| \\
\circ & \bigcirc_{C \in \mathcal{C}_k} |\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R|
\end{aligned}$$

- **Claim:** There is a TAG that generates

$$\{\mathrm{GG_k}(G) \mid G \text{ contains a } 4k\text{-clique}\}$$
$$\cup \{\text{some strings that are not encodings of graphs}\}$$

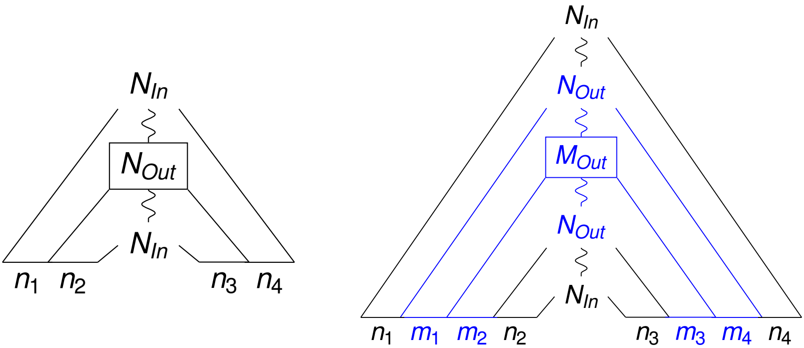
Reminder: Tree-Adjoining Grammars

- Initial and auxiliary trees, nodes marked for adjunction



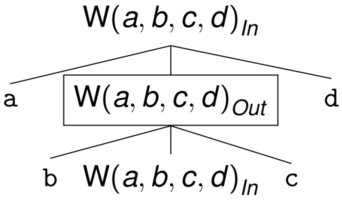
Hardness for Tree-Adjoining Grammars

- Consider sets of special trees ⇝ *Program*
- Easily combinable



Hardness for Tree-Adjoining Grammars

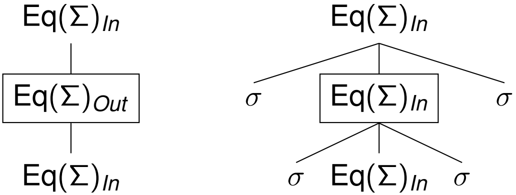
Writing characters $\mathrm{W}(a, b, c, d)$



Generates (a, b, c, d)

Hardness Result

Testing for equality Eq(Σ)

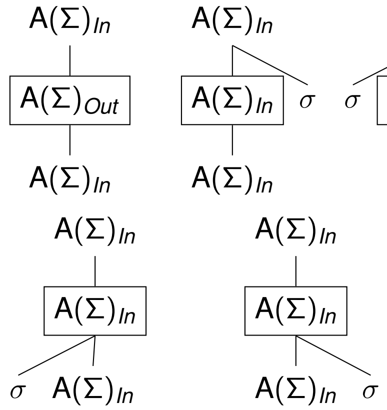


$\forall \sigma \in \Sigma$

Generates $\{(s, s^R, s, s^R) \mid s \in \Sigma^*\}$

Hardness for Tree-Adjoining Grammars

Writing anything $A(\Sigma)$

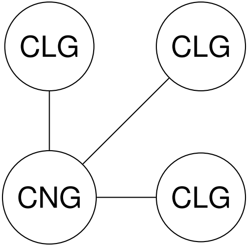


$\forall \sigma \in \Sigma$

Generates $\Sigma^* \times \Sigma^* \times \Sigma^* \times \Sigma^*$

Programs

- Combine programs to detect claws of cliques



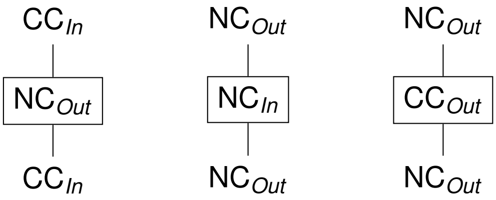
- Detect $4k$-clique as 4 claws of cliques

Hardness for Tree-Adjoining Grammars

Detecting claws

$$\begin{aligned}\mathrm{NC} := {} & \mathrm{W}(\#) \cdot \mathrm{A}(\{0,1,\$\}) \cdot \mathrm{W}(\$) \\ & \cdot \mathrm{Eq}(\{0,1\}) \\ & \cdot \mathrm{W}(\$) \cdot \mathrm{A}(\{0,1,\$\}) \cdot \mathrm{W}(\#)\end{aligned}$$

Detecting claws of k-cliques (applying NC k^2 times)



- CC is completely symmetric
- Can simply use it 4 times:

$$\begin{aligned}\mathrm{C} := {} & \mathrm{A}(0,1,\#,\S,|)\cdot \mathrm{W}(|)\\ & \cdot \mathrm{CC}\cdot \mathrm{W}(\S)\cdot \mathrm{CC}\cdot \mathrm{W}(\S)\cdot \mathrm{CC}\cdot \mathrm{W}(\S)\cdot \mathrm{CC}\\ & \cdot \mathrm{W}(|)\cdot \mathrm{A}(0,1,\#,\S,|)\end{aligned}$$

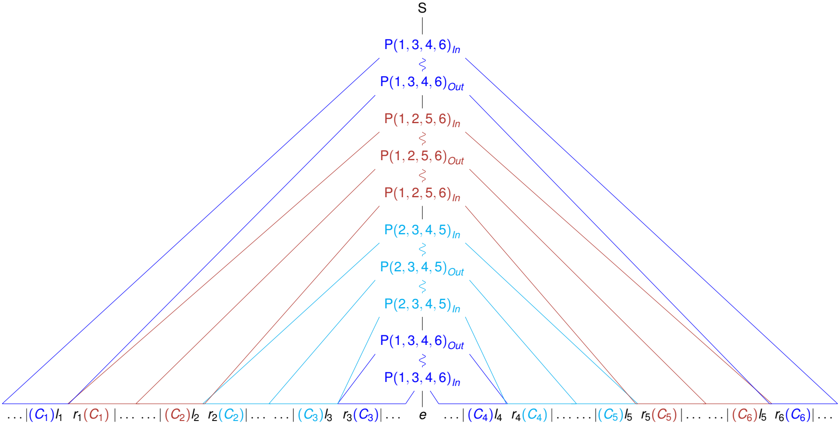
- $$\begin{aligned}\mathrm{GG_k}(G) := {} & \bigcirc_{C\in\mathcal{C}_k} |\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R|\\ & \circ \bigcirc_{C\in\mathcal{C}_k} |\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R|\\ & \circ \bigcirc_{C\in\mathcal{C}_k} |\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R|\\ & \circ \bigcirc_{C\in\mathcal{C}_k} |\mathrm{CNG}(C)\S\mathrm{CLG}(C)^R\S\mathrm{CLG}(C)\S\mathrm{CLG}(C)^R|\end{aligned}$$

Conclusions

- Current TAG parsers are very likely optimal
- There is a natural problem with the running time
  - $\Theta(n^{2\omega})$ using matrix multiplication
  - $\Theta(n^6)$ not using matrix multiplication

More Details for TAG-hardness

- Overview of the grammar parsing $6k$-cliques



More Details for TAG-hardness

Encoding of the graph, the string s

$$
\begin{aligned}
\mathrm{GG}_k(G) :=\ & \bigcirc_{C\in\mathcal{C}_k} \mid \mathrm{CNG}(C)\ \S\ \mathrm{CLG}(C)^R\ l_1\ r_1\ \mathrm{CLG}(C)\ \S\ \mathrm{CLG}(C)^R \mid \\
\circ & \bigcirc_{C\in\mathcal{C}_k} \mid \mathrm{CNG}(C)\ \S\ \mathrm{CLG}(C)^R\ l_2\ r_2\ \mathrm{CLG}(C)\ \S\ \mathrm{CLG}(C)^R \mid \\
\circ & \bigcirc_{C\in\mathcal{C}_k} \mid \mathrm{CNG}(C)\ \S\ \mathrm{CLG}(C)^R\ l_3\ r_3\ \mathrm{CLG}(C)\ \S\ \mathrm{CLG}(C)^R \mid \\
\circ\ & e \\
\circ & \bigcirc_{C\in\mathcal{C}_k} \mid \mathrm{CLG}(C)\ \S\ \mathrm{CLG}(C)^R\ l_4\ r_4\ \mathrm{CNG}(C)\ \S\ \mathrm{CLG}(C)^R \mid \\
\circ & \bigcirc_{C\in\mathcal{C}_k} \mid \mathrm{CLG}(C)\ \S\ \mathrm{CLG}(C)^R\ l_5\ r_5\ \mathrm{CNG}(C)\ \S\ \mathrm{CLG}(C)^R \mid \\
\circ & \bigcirc_{C\in\mathcal{C}_k} \mid \mathrm{CLG}(C)\ \S\ \mathrm{CLG}(C)^R\ l_6\ r_6\ \mathrm{CNG}(C)\ \S\ \mathrm{CLG}(C)^R \mid
\end{aligned}
$$

References I

- Christy Doran, Dania Egedi, Beth Ann Hockey, Bangalore Srinivas, and Martin Zaidel, *XTAG system: a wide coverage grammar for English*, 15th Conference on Computational Linguistics, COLING'94, 1994, pp. 922–928.
- Jaroslav Nešetřil and Svatopluk Poljak, *On the complexity of the subgraph problem*, Commentationes Mathematicae Universitatis Carolinae **26** (1985), no. 2, 415–419.
- Sanguthevar Rajasekaran and Shibu Yooseph, *TAL recognition in $O(M(N^2))$ time*, JCSS **56** (1998), no. 1, 83–89.

References II

- Giorgio Satta, *Tree-adjoining Grammar Parsing and Boolean Matrix Multiplication*, Comput. Linguist. **20** (1994), no. 2, 173–191.
- Yves Schabes and Aravind K. Joshi, *An Earley-type parsing algorithm for tree adjoining grammars*, 26th Annual Meeting of the Association for Computational Linguistics, ACL'88, 1988, pp. 258–269.
- K. Vijay-Shankar and Aravind K. Joshi, *Some computational properties of tree adjoining grammars*, 23rd Annual Meeting of the Association for Computational Linguistics, ACL'85, 1985, pp. 82–93.

Navigation

Start