

---

# Efficient Factorization of Synchronous Context-Free Grammars

Hao Zhang and Daniel Gildea

Computer Science Department

University of Rochester

---

## Outline

- Introduction to Synchronous Context Free Grammar (SCFG)
- A shift-reduce algorithm for factorization of SCFG
- An inductive proof for the correctness of the algorithm

## Synchronous CFG vs. CFG

	CFG	SCFG									
terminals	words	word pairs									
nonterminals	phrases	phrase pairs									
productions	sequences  $X \rightarrow X_1X_2X_3$	<i>permutations</i>  $X \rightarrow$ <table border="1" style="display: inline-table; vertical-align: middle; text-align: center;"> <tr><td></td><td></td><td><math>X_3</math></td></tr> <tr><td><math>X_1</math></td><td></td><td></td></tr> <tr><td></td><td><math>X_2</math></td><td></td></tr> </table>			$X_3$	$X_1$				$X_2$	
		$X_3$									
$X_1$											
	$X_2$										
parsing	over spans	over cells									

---

## Notions for SCFG

- A generic n-ary SCFG rule is written as

$$X \rightarrow X_1^{(1)} \dots X_n^{(n)}, X_{\pi(1)}^{(\pi(1))} \dots X_{\pi(n)}^{(\pi(n))}$$

where each  $X_i$  is a variable which can take the value of any nonterminal in the grammar.

- For example, the 3-ary rule  $S \rightarrow$

		A
A		
	B	

can be written

as

$$S \rightarrow A^{(1)} B^{(2)} A^{(3)}, B^{(2)} A^{(1)} A^{(3)}$$

where  $\pi = (2, 1, 3)$ .

---

## Parsing Complexity of n-ary SCFG

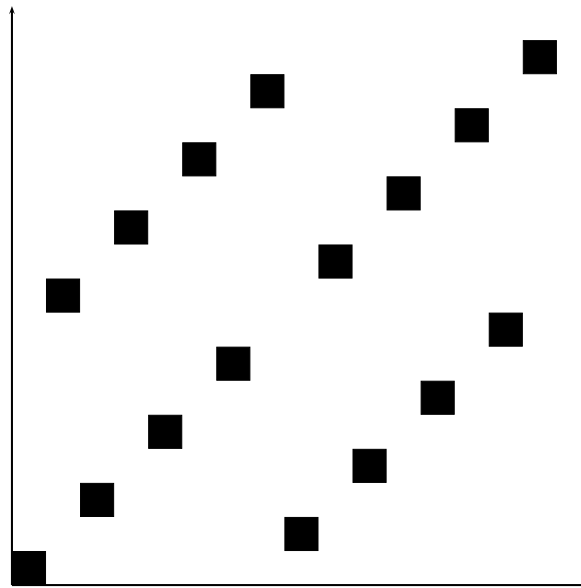
Assume the sentence length (on either side) is  $O(N)$ .

- Naïve parsing is  $O(N^{2n+2})$ , by enumerating at most  $2(n + 1)$  split points into the two sentences.
- $O(N^{n+1+3}) = O(N^{n+4})$  is also doable, if we go from left to right on one side and pick up one pebble a time, allowing discontinuous spans on the other side.
- But even if the strategy allows discontinuous spans on both sides, it is  $\Omega(N^{c\sqrt{n}})$ , (Satta and Peserico, 2005), because there exist hard-for-parsing permutations out of  $n!$ .

---

## A Hard-for-Parsing Permutation

If you imagine the  $n$  pebbles are spread on the  $n$  by  $n$  plane as far away from one another as possible, you can get an idea of the extreme case.



---

## Parsing Complexity and Branching Factor

Compare n-ary CFG and n-ary SCFG

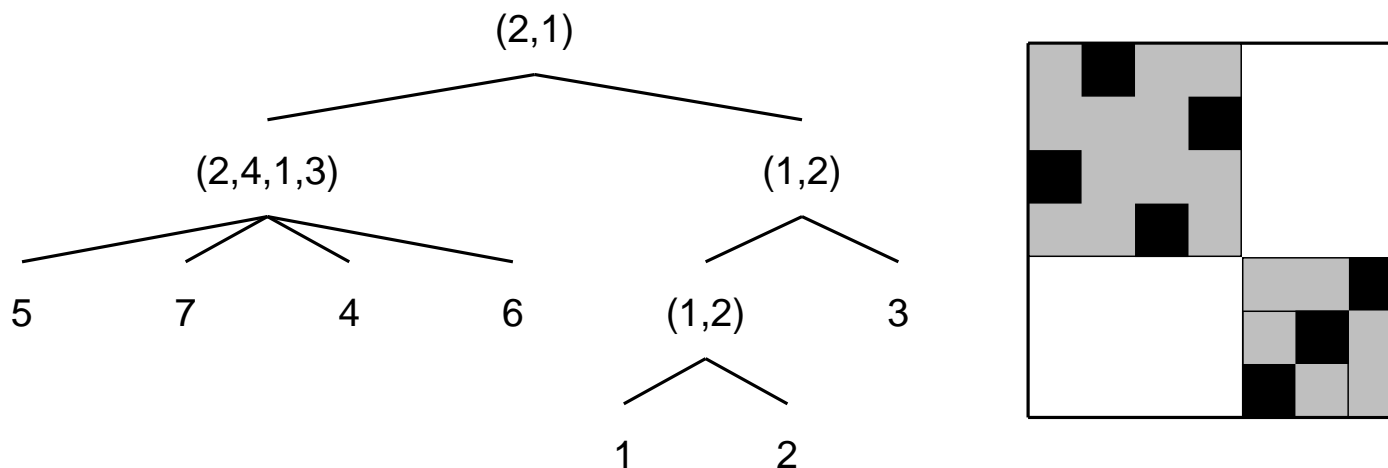
	CFG	SCFG
n = 2	$O(N^3)$	$(N^6)$
large n	$O(N^3)$	$\Omega(N^{c\sqrt{n}})$ and $O(N^{n+4})$

- Why so? One possible structure vs.  $n!$  structures. Different permutation patterns lead to different degrees of difficulty in factorization.

---

## Our Approach to Reducing SCFG Parsing Complexity

Not all permutations are hard-for-parsing (difficult-to-factorize). For example, (5, 7, 4, 6, 1, 2, 3) is decomposable:



So we can reduce long SCFG rules into shorter ones. If the longest SCFG rule after such factorization is  $k$ , we can parse sentences with the grammar in  $O(N^{k+4})$ .

---

## Notions for Parsing Permutations

- **permuted sequence**: such as (5, 7, 4, 6, 1, 2, 3), (5, 7, 4, 6), and (1, 2). If a permuted sequence has been found, we can reduce it to a subsequence (block) of [min...max], such as [4...7] and [1...2]. A block serves as a pebble in latter reductions.
- **permutation tree**: a hierarchy of permuted sequences.
- **k-arizer**: parse a permutation into a permutation tree with the maximal fanout of any node as k.

---

## Shift-Reduce k-arizer

1. Shift the next number in the input permutation onto the stack.
2. Repeatedly try the 2-ary, 3-ary, ..., k-ary permutations to reduce the subsequences on the top of the stack to one long subsequence.
3. If there are remaining numbers in the input permutation, go to 1.

---

## An Example

Stack	Input	
	5, 7, 4, 6, 1, 2, 3	
5, 7	4, 6, 1, 2, 3	shift
5, 7, 4	6, 1, 2, 3	shift
5, 7, 4, 6	1, 2, 3	shift
[4... 7]	1, 2, 3	reduce by (2,4,1,3)
[4... 7], [1... 2]	3	reduce by (1,2)
[4... 7], [1... 3]		reduce by (1,2)
[1... 7]		reduce by (2,1)

---

## Max-Min Checking for Linear-time Reduction

Let's focus on the reduction over (5, 7, 4, 6):

Stack	Max-Min	Depth	Action
5, 7, [4, 6]	$ [4...6]  = 3 \neq$	2	Go deeper
5, [7, 4, 6]	$ [4...7]  = 4 \neq$	3	Go deeper
[5, 7, 4, 6]	$ [4...7]  = 4 =$	4	Reduce

- Each reduction takes  $O(k)$ . The entire algorithm is  $O(nk)$ .

---

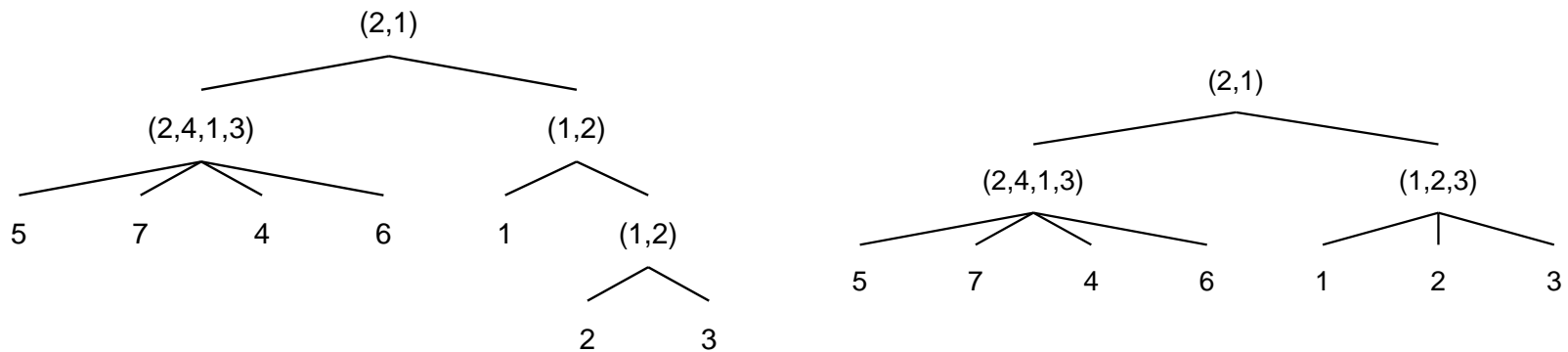
## Arguments for Correctness

- If the algorithm succeeds on an input permutation, we have a k-ary permutation tree.
- On the other hand, if a permutation has one or more k-ary parse trees, can the k-arizer find any of them?

---

## Normalized Permutation Tree

Actually,  $(5, 7, 4, 6, 1, 2, 3)$ ,  $(5, 7, 4, 6)$  has two other 4-ary parse trees. They can be easily transformed into the left-heavy minimal tree we have seen before.



- We will show that the actions of the k-arizer will correspond one-to-one with the left-most derivation of the normalized permutation tree.

---

## Proof by Induction

- Suppose till the  $i$ -th reduction ( $i \geq 0$ ), the  $k$ -arizer always chooses the same reduction as indicated by the reference.
- We enumerate all of the possible cases of mismatching over the next-should-be-reduced subsequences and show they are all contradictory cases.

---

## A Case That Requires Analysis

When the reduction of the reference and the reduction of the k-arizer mutually overlap:

$$S_1 S_2 \dots [S_p \dots \overbrace{S_{p+x} \dots S_q}^{r'_j} \dots S_{q+y}]$$

Consider

- when there are more than one blocks in the overlapping region
- when there are more than one blocks in either of the non-overlapping regions
- the third case, a truly ambiguous case, like (1, (2), 3):

$$S_1 S_2 \dots [S_p \overbrace{S_q}^{r'_j} S_{q+1}]$$

---

## Conclusions

- We can efficiently produce a unique normalized minimal parse tree for each permutation.
- If the fanout of the normalized tree is  $k$ , let us call the permutation  $k$ -arizable.
- When a permutation of  $n$  is only  $n$ -arizable, our  $k$ -arizer runs in  $O(n^2)$ .
- For SCFG factorization, we decorate the permutation tree with nonterminals and read off the rules from the tree.

---

## Further Questions

- Can we have an  $O(n \log(n))$  factorization algorithm?  
Yes. (Satta, Gildea, and Zhang, submitted)
- Can we have an  $O(n)$  algorithm?  
We don't know.
- How many permutations of  $n$  are  $k$ -arizable?  
To be answered next time.