

Implementation of PCFG parsers

Practical Course NLP

Boris Petrov

Chair for Algorithmics
Institute for theoretical computer science
TU Dresden

May 12th

Outline

- 1 Overview
- 2 CKY parsing
- 3 Deductive parsing
- 4 Conclusion

Overview

Parser invocation:

```
./pcfg_tool parse grammar.rules grammar.lexicon < sentences
```

Parser mainloop:

read PCFG G from files

G ↓
read sentence w (1 line) from `stdin`

G, w ↓
weighted CKY parsing or deductive parsing

print best derivation tree

d ↓

Outline

- 1 Overview
- 2 CKY parsing
- 3 Deductive parsing
- 4 Conclusion

The CKY parsing algorithm (formal notation)

Require: CFG $G = (N, \Sigma, R, S)$ in CNF, sentence $w = w_1 \dots w_n$ where $w_1, \dots, w_n \in \Sigma$

Ensure: family of sets $(c_{i,j} \subseteq N \mid 0 \leq i < j \leq n)$ such that $A \in c_{i,j} \iff D_G^A(w_{i+1} \dots w_j) \neq \emptyset$

```
1: function CKY( $R, w_1 \dots w_n$ )
2:   for  $i := 1$  to  $n$  do
3:      $c_{i-1,i} := \{A \mid A \rightarrow w_i \in R\}$ 
4:   for  $r := 2$  to  $n$  do
5:     for  $i := 0$  to  $n - r$  do
6:        $j := i + r$ 
7:        $c_{i,j} := \{A \mid i < m < j, A \rightarrow BC \in R: B \in c_{i,m}, C \in c_{m,j}\}$ 
8:   return  $(c_{i,j} \mid 0 \leq i < j \leq n)$ 
```

where

- CNF = Chomsky normal form
- $D_G^A(w_i \dots w_j)$ = set of (left) derivations of A in G that result in $w_i \dots w_j$
- $c_{i,j}$ = set of all nonterminals that derive $w_{i+1} \dots w_j$

The CKY parsing algorithm (imperative flavor)

Require: CFG $G = (N, \Sigma, R, S)$ in CNF, sentence $w = w_1 \dots w_n$ where $w_1, \dots, w_n \in \Sigma$

Ensure: family of sets $(c_{i,j} \subseteq N \mid 0 \leq i < j \leq n)$ such that $A \in c_{i,j} \iff D_G^A(w_{i+1} \dots w_j) \neq \emptyset$

```
1: function CKY( $R, w_1 \dots w_n$ )
2:   initialize  $c$  ▷  $c_{i,j} := \emptyset$  for each  $0 \leq i < j \leq n$ 
3:   for  $i := 1$  to  $n$  do
4:     for  $A \rightarrow w_i \in R$  do
5:        $c_{i-1,i} := c_{i-1,i} \cup \{A\}$  }  $c_{i-1,i} := \{A \mid A \rightarrow w_i \in R\}$ 
6:   for  $r := 2$  to  $n$  do ▷ length of span
7:     for  $i := 0$  to  $n - r$  do ▷ start of span
8:        $j := i + r$  ▷ end of span
9:       for  $A \in N$  do
10:        for  $m := i + 1$  to  $j - 1$  do ▷ partition position
11:          for  $A \rightarrow BC \in R$  do
12:            if  $B \in c_{i,m}$  and  $C \in c_{m,j}$  then }  $c_{i,j} := \{A \mid i < m < j, A \rightarrow BC \in$ 
13:               $c_{i,j} := c_{i,j} \cup \{A\}$  }  $R: B \in c_{i,m}, C \in c_{m,j}\}$ 
14:   return  $(c_{i,j} \mid 0 \leq i < j \leq n)$ 
```

From unweighted to weighted CKY parsing

	Unweighted	Weighted
data structure for spans	set of nonterminals $(c_{i,j} \subseteq N \mid 0 \leq i < j \leq n)$	map nonterminals to real numbers $(c_{i,j}: N \rightarrow \mathbb{R} \mid 0 \leq i < j \leq n)$
update operation for rule $r = A \rightarrow \dots$	$c_{i,j} := c_{i,j} \cup \{A\}$	$c_{i,j}(A) := \max\{c_{i,j}(A), p(r)\}$

Convention: we will write $c_{i,j,A}$ rather than $c_{i,j}(A)$

The CKY parsing algorithm + weights

Require: PCFG (N, Σ, R, S, p) in CNF, sentence $w = w_1 \dots w_n$ where $w_1, \dots, w_n \in \Sigma$

Ensure: family $(c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)$ such that

$$c_{i,j,A} = \max (\{p(d) \mid d \in D_G^A(w_{i+1} \dots w_j)\} \cup \{0\})$$

```
1: function CKY( $R, p, w_1 \dots w_n$ )
2:   initialize  $c$ 
3:   for  $i := 1$  to  $n$  do
4:     for  $A \rightarrow w_i \in R$  do
5:        $c_{i-1,i,A} := \max\{c_{i-1,i,A}, p(A \rightarrow w_i)\}$ 
6:   for  $r := 2$  to  $n$  do
7:     for  $i := 0$  to  $n - r$  do
8:        $j := i + r$ 
9:       for  $A \in N$  do
10:        for  $m := i + 1$  to  $j - 1$  do
11:          for  $A \rightarrow BC \in R$  do
12:             $c_{i,j,A} := \max\{c_{i,j,A}, p(A \rightarrow BC) \cdot c_{i,m,B} \cdot c_{m,j,C}\}$ 
13:   return  $(c_{i,j,A} \mid 0 \leq i < j \leq n, A \in N)$ 
```

$\triangleright c_{i,j,A} := 0$ for each $0 \leq i < j \leq n$ and $A \in N$

Adding chain rules

Types of rules accepted by classic CKY:

$$A \rightarrow w$$

(with $A, B, C \in N$ and $w \in \Sigma$)

$$A \rightarrow BC$$

What we also want:

$$A \rightarrow B$$

(with $A, B \in N$)

Solution: two-phase procedure per cell $c_{i,j}$

- 1 perform classic CKY step
- 2 compute closure under unary rules

The CKY parsing algorithm + weights + chain rules

Require: binary PCFG (N, Σ, R, S, p) , sentence $w = w_1 \dots w_n$ where $w_1, \dots, w_n \in \Sigma$

Ensure: family $(c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)$ such that $c_{i,j,A} = \max(\{p(d) \mid d \in D_G^A(w_{i+1} \dots w_j)\} \cup \{0\})$

```
1: function CKY( $R, \mu, w_1 \dots w_n$ )
2:   initialize  $c$  ▷  $c_{i,j,A} := 0$  for each  $0 \leq i < j \leq n$  and  $A \in N$ 
3:   for  $i := 1$  to  $n$  do
4:     for  $A \rightarrow w_i \in R$  do
5:        $c_{i-1,i,A} := p(A \rightarrow w_i)$ 
6:        $(c_{i-1,i,A} \mid A \in N) = \text{UNARY\_CLOSURE}(R, p, (c_{i-1,i,A} \mid A \in N))$ 
7:   for  $r := 2$  to  $n$  do
8:     for  $i := 0$  to  $n - r$  do
9:        $j := i + r$ 
10:      for  $A \in N$  do
11:        for  $m := i + 1$  to  $j - 1$  do
12:          for  $A \rightarrow BC \in R$  do
13:             $c_{i,j,A} := \max\{c_{i,j,A}, p(A \rightarrow BC) \cdot c_{i,m,B} \cdot c_{m,j,C}\}$ 
14:             $(c_{i,j,A} \mid A \in N) = \text{UNARY\_CLOSURE}(R, p, (c_{i,j,A} \mid A \in N))$ 
15:   return  $(c_{i,j,A} \mid 0 \leq i < j \leq n, A \in N)$ 
```

The CKY parsing algorithm + weights + chain rules

```
16: function UNARY_CLOSURE( $R, p, (c_A \mid A \in N)$ )
17:    $queue := \{(A, c_A) \mid A \in N, c_A \neq 0\}$ 
18:   for  $A \in N$  do
19:      $c_A := 0$ 
20:   while  $queue \neq \emptyset$  do
21:      $(B, q) := \operatorname{argmax}_{(\hat{B}, \hat{q}) \in queue} \hat{q}$ 
22:      $queue := queue \setminus \{(B, q)\}$ 
23:     if  $c_B < q$  then
24:        $c_B := q$ 
25:       for  $A \rightarrow B \in R$  do
26:          $queue := queue \cup \{(A, p(A \rightarrow B) \cdot q)\}$ 
27:   return  $(c_A \mid A \in N)$ 
```

About backtraces

What we have: best weight $c_{i,j,A}$ for derivations in $D_G^A(w_{i+1} \dots w_j)$ for each $A \in N$, $0 \leq i < j \leq n$

What we want: the best derivation of S that results in $w = w_1 \dots w_n$

During the CKY algorithm:

- store *backtraces* (indicators *how* a weight was computed)
- for best derivation: at most one backtrace per span and nonterminal
- update when weight is updated

After the CKY algorithm:

- recursively read trees from backtraces:

Require: family of backtraces b , each otf. \perp , or $A \rightarrow t$, or $(A \rightarrow B, i, j)$ or $(A \rightarrow BC, i, m, m, j)$

- 1: **function** BEST_TREE($(b_{i,j,A} \mid 0 \leq i < j \leq n, A \in N), i, j, A$)
- 2: **if** $b_{i,j,A}$ otf. $A \rightarrow t$ **then return** $A \rightarrow t$
- 3: **else if** $b_{i,j,A}$ otf. $(A \rightarrow B, i, j)$ **then return** $(A \rightarrow B)(\text{BEST_TREE}(b, i, j, B))$
- 4: **else if** $b_{i,j,A}$ otf. $(A \rightarrow BC, i, m, m, j)$ **then**
- 5: **return** $(A \rightarrow BC)(\text{BEST_TREE}(b, i, m, B), \text{BEST_TREE}(b, m, j, C))$

Let's talk about data structures

- access to grammar rules depends on loops:
 - access by first nonterminal on rhs
 - for some, that be no concern

```
Map<Nt, Set<(Rule, Wt)>>  
    Set<(Rule, Wt)>
```

Let's talk about data structures

- access to grammar rules depends on loops:

- access by first nonterminal on rhs
- for some, that be no concern

Map<Nt, Set<(Rule, Wt)>>
Set<(Rule, Wt)>

- weights for each span and nonterminal:

- usually in a $(\frac{|w| \cdot (|w|+1)}{2} \cdot |N|)$ -dimensional vector (dense)
- or hashmap (sparse)

Vec<Wt>
Map<(Int, Int, Nt), Wt>

Let's talk about data structures

- access to grammar rules depends on loops:

- access by first nonterminal on rhs
- for some, that be no concern

Map<Nt, Set<(Rule, Wt)>>
Set<(Rule, Wt)>

- weights for each span and nonterminal:

- usually in a $(\frac{|w| \cdot (|w|+1)}{2} \cdot |N|)$ -dimensional vector (dense)
- or hashmap (sparse)

Vec<Wt>
Map<(Int, Int, Nt), Wt>

- storing backtraces:

- each backtrace: rule and spans and nonterminals (on rule's right-hand side)

Bt = Bin(Rule, [Int; 4])+ Chain(Rule, [Int; 2])+ Term(Rule)

- one backtrace for each span and nonterminal (dense)
- or in a hashmap (sparse)
- or do not store them at all

Vec<Bt>
Map<(Int, Int, Nt), Bt>

Outline

- 1 Overview
- 2 CKY parsing
- 3 Deductive parsing**
- 4 Conclusion

Deduction systems [Ned03]

- rule-based system (derivation of *items*)
- derive consequence (c) from antecedents (a_1, \dots, a_k) for some $k \in \mathbb{N}$

$$\frac{a_1, \dots, a_k}{c}$$

Deduction systems [Ned03]

- rule-based system (derivation of *items*)
- derive consequence (c) from antecedents (a_1, \dots, a_k) for some $k \in \mathbb{N}$
- compute weight of consequence using weight of antecedents (q_1, \dots, q_k)

$$\frac{a_1: q_1, \dots, a_k: q_k}{c: f(q_1, \dots, q_k)}$$

Deduction systems [Ned03]

- rule-based system (derivation of *items*)
- derive consequence (c) from antecedents (a_1, \dots, a_k) for some $k \in \mathbb{N}$
- compute weight of consequence using weight of antecedents (q_1, \dots, q_k)
- side condition b

$$\frac{a_1: q_1, \dots, a_k: q_k}{c: f(q_1, \dots, q_k)} b$$

Deduction system for parsing weighted CFG [Ned03]

- item (i, A, j) for each nonterminal A spanning $w_{i+1} \dots w_j$

Deduction system for parsing weighted CFG [Ned03]

- item (i, A, j) for each nonterminal A spanning $w_{i+1} \dots w_j$
- predict initial items $\frac{}{(i-1, A, i): p(A \rightarrow w_i)} A \rightarrow w_i \in R \wedge W = w_1 \dots w_i \dots w_n$

Deduction system for parsing weighted CFG [Ned03]

- item (i, A, j) for each nonterminal A spanning $w_{i+1} \dots w_j$
- predict initial items $\frac{}{(i-1, A, i): p(A \rightarrow w_i)} A \rightarrow w_i \in R \wedge W = w_1 \dots w_i \dots w_n$
- combine items $\frac{(i_0, B_1, i_1): q_1, (i_1, B_2, i_2): q_2, \dots, (i_{k-1}, B_k, i_k): q_k}{(i_0, A, i_k): p(A \rightarrow B_1 \dots B_k) \cdot q_1 \dots q_k} A \rightarrow B_1 \dots B_k \in R$

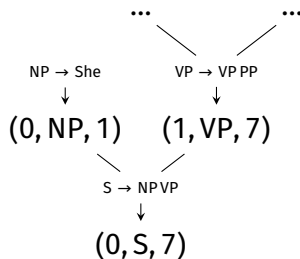
Deduction system for parsing weighted CFG [Ned03]

- item (i, A, j) for each nonterminal A spanning $w_{i+1} \dots w_j$
- predict initial items $\frac{}{(i-1, A, i): p(A \rightarrow w_i)} A \rightarrow w_i \in R \wedge W = w_1 \dots w_i \dots w_n$
- combine items $\frac{(i_0, B_1, i_1): q_1, (i_1, B_2, i_2): q_2, \dots, (i_{k-1}, B_k, i_k): q_k}{(i_0, A, i_k): p(A \rightarrow B_1 \dots B_k) \cdot q_1 \dots q_k} A \rightarrow B_1 \dots B_k \in R$
- goal item: $(0, S, |w|)$

Deduction system for parsing weighted CFG [Ned03]

- item (i, A, j) for each nonterminal A spanning $w_{i+1} \dots w_j$
- predict initial items $\frac{}{(i-1, A, i): p(A \rightarrow w_i)} A \rightarrow w_i \in R \wedge W = w_1 \dots w_i \dots w_n$
- combine items $\frac{(i_0, B_1, i_1): q_1, (i_1, B_2, i_2): q_2, \dots, (i_{k-1}, B_k, i_k): q_k}{(i_0, A, i_k): p(A \rightarrow B_1 \dots B_k) \cdot q_1 \dots q_k} A \rightarrow B_1 \dots B_k \in R$
- goal item: $(0, S, |w|)$

- deduction system \rightsquigarrow weighted hypergraph
 - edge from antecedents to consequence
 - can be explored with respect weight
 - hyperpaths to goal item correspond to parse trees



Weighted deductive parsing algorithm

Require: weighted binary cfg (N, Σ, R, S, p) , word $w_1 \dots w_n$ where $w_1, \dots, w_n \in \Sigma$

Ensure: family $(c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)$ such that $c_{i,j,A} = \max(\{p(d) \mid d \in D_G^A(w_i \dots w_j)\} \cup \{0\})$

```
1: function DEDUCE( $R, p, w_1 \dots w_n$ )
2:    $queue := \{(i-1, A, i, p(A \rightarrow w_i)) \mid 1 \leq i \leq n, A \rightarrow w_i \in R\}$ 
3:   initialize  $c$   $\triangleright c_{i,j,A} := 0$  for each  $0 \leq i < j \leq n$  and  $A \in N$ 
4:   while  $queue \neq \emptyset$  do
5:      $(i, A, j, q) := \operatorname{argmax}_{(\hat{i}, \hat{A}, \hat{j}, \hat{q}) \in queue} \hat{q}$ 
6:      $queue := queue \setminus \{(i, A, j, q)\}$ 
7:     if  $c_{i,j,A} = 0$  then
8:        $c_{i,j,A} := q$ 
9:        $queue := queue \cup \{(i, A', j', p(A' \rightarrow AC) \cdot q \cdot c_{j,j',C}) \mid j < j' \leq n, A' \rightarrow AC \in R\}$ 
10:       $queue := queue \cup \{(i', A', j, p(A' \rightarrow BA) \cdot c_{i',j,B} \cdot q) \mid 0 \leq i' < i, A' \rightarrow BA \in R\}$ 
11:       $queue := queue \cup \{(i, A', j, p(A' \rightarrow A) \cdot q) \mid A' \rightarrow A \in R\}$ 
12:   return  $(c_{i,j,A} \mid 0 \leq i < j \leq n, A \in N)$ 
```

Let's talk about data structures ... again

- access of grammar from each rhs nonterminal

Map<Nt, Set<(Rule, Wt)>>

Let's talk about data structures ... again

- access of grammar from each rhs nonterminal
- each item may need to store a backtrace

```
Map<Nt, Set<(Rule, Wt)>>  
(Int, Nt, Int, Wt, Bt)
```

Let's talk about data structures ... again

- access of grammar from each rhs nonterminal `Map<Nt, Set<(Rule, Wt)>>`
- each item may need to store a backtrace `(Int, Nt, Int, Wt, Bt)`
- storing the found items and their weights:
 - access from left `Map<(Int, Nt), Set<(Int, Nt, Int, Wt)>>`
 - access from right `Map<(Nt, Int), Set<(Int, Nt, Int, Wt)>>`

Let's talk about data structures ... again

- access of grammar from each rhs nonterminal `Map<Nt, Set<(Rule, Wt)>>`
- each item may need to store a backtrace `(Int, Nt, Int, Wt, Bt)`
- storing the found items and their weights:
 - access from left `Map<(Int, Nt), Set<(Int, Nt, Int, Wt)>>`
 - access from right `Map<(Nt, Int), Set<(Int, Nt, Int, Wt)>>`
- storing backtraces:
 - store applied rule and antecedent items
`Bt = Bin(Rule, [Int; 4])+ Chain(Rule, [Int; 2])+ Term(Rule)`
 - one backtrace for each item `Map<(Int, Nt, Int), Bt>`
 - or do not store them at all

Outline

- 1 Overview
- 2 CKY parsing
- 3 Deductive parsing
- 4 Conclusion**

General Comments and Tips

- order of loops in CKY algorithm doesn't matter that much, *but*¹:
 - may be used to cache-optimize,
 - may lead to other optimizations

¹Bodenstab [Bod09] discusses this in detail.

General Comments and Tips

- order of loops in CKY algorithm doesn't matter that much, *but*¹:
 - may be used to cache-optimize,
 - may lead to other optimizations
- deductive parsers may not need to expand the whole search space

¹Bodenstab [Bod09] discusses this in detail.

General Comments and Tips

- order of loops in CKY algorithm doesn't matter that much, *but*¹:
 - may be used to cache-optimize,
 - may lead to other optimizations
- deductive parsers may not need to expand the whole search space
- try to think about efficient access in your data structures
 - don't search in lists
 - indexed access: maps
 - check if you *really* need sets/maps
 - flat data structures are faster than stacked heap allocations

¹Bodenstab [Bod09] discusses this in detail.

General Comments and Tips

- order of loops in CKY algorithm doesn't matter that much, *but*¹:
 - may be used to cache-optimize,
 - may lead to other optimizations
- deductive parsers may not need to expand the whole search space
- try to think about efficient access in your data structures
 - don't search in lists
 - indexed access: maps
 - check if you *really* need sets/maps
 - flat data structures are faster than stacked heap allocations
- nonterminals: work with integers rather than strings

¹Bodenstab [Bod09] discusses this in detail.

General Comments and Tips

- order of loops in CKY algorithm doesn't matter that much, *but*¹:
 - may be used to cache-optimize,
 - may lead to other optimizations
- deductive parsers may not need to expand the whole search space
- try to think about efficient access in your data structures
 - don't search in lists
 - indexed access: maps
 - check if you *really* need sets/maps
 - flat data structures are faster than stacked heap allocations
- nonterminals: work with integers rather than strings
- try not to over-engineer it

¹Bodenstab [Bod09] discusses this in detail.

- [Bod09] Nathan Bodenstab. “Efficient Implementation of the cky algorithm”. *Computational Linguistics, Final Project Paper*, 2009.
- [CS70] John Cocke and J. T. Schwartz. *Programming languages and their compilers: Preliminary notes*. Tech. rep. Version 2nd. Courant Institute of Mathematical Sciences, New York University, 1970.
- [HC05] Liang Huang and David Chiang. “Better k-best parsing”. *Association for Computational Linguistics*. 2005.
- [Kas66] T. Kasami. *An efficient recognition and syntax-analysis algorithm for context-free languages*. Tech. rep. AFCRL, 1966.
- [Ned03] Mark-Jan Nederhof. “Weighted deductive parsing and Knuth’s algorithm”. *Computational Linguistics*, 2003.
- [You67] Daniel H. Younger. “Recognition and parsing of context-free languages in time n^3 ”. *Information and Control*, 1967.