Implementation of PCFG parsers

Praktikum NLP

Thomas Ruprecht Richard Mörbitz

Chair for foundations of programming
Institute for theoretical computer science
TU Dresden

May 10th

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

The CKY parsing algorithm (formal notation)

Require: CFG $G = (N, \Sigma, R, S)$ in CNF, sentence $w = w_1 \ldots w_n$ where $w_1, \ldots, w_n \in \Sigma$
Ensure: family of sets $(c_{i,j} \subseteq N \mid 0 \leq i < j \leq n)$ such that $A \in c_{i,j} \iff D_G^A(w_{i+1} \ldots w_j) \neq \emptyset$

function CKY($R, w_1 \ldots w_n$)
  for $i := 1$ **to** n **do**
    $c_{i-1,i} := \{A \mid A \to w_i \in R\}$
  for $r := 2$ **to** n **do**
    for $i := 0$ **to** $n - r$ **do**
      $j := i + r$
      $c_{i,j} := \{A \mid i < m < j, A \to BC \in R \colon B \in c_{i,m}, C \in c_{m,j}\}$
  return $(c_{i,j} \mid 0 \leq i < j \leq n)$

where

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

The CKY parsing algorithm (imperative flavor)

Require: CFG $G = (N, \Sigma, R, S)$ in CNF, sentence $w = w_1 \ldots w_n$ where $w_1, \ldots, w_n \in \Sigma$
Ensure: family of sets $(c_{i,j} \subseteq N \mid 0 \le i < j \le n)$ such that $A \in c_{i,j} \iff D_G^A(w_{i+1} \ldots w_j) \neq \emptyset$

function CKY($R, w_1 \ldots w_n$)
    $(c_{i,j} := \emptyset \mid 0 \le i < j \le n)$
    for $i := 1$ **to** n **do**
        for $A \to w_i \in R$ **do**
            $c_{i-1,i} := c_{i-1,i} \cup \{A\}$

$c_{i-1,i} := \{A \mid A \to w_i \in R\}$

    for $r := 2$ **to** n **do** ▷ length of span
        for $i := 0$ **to** $n - r$ **do** ▷ start of span
            $j := i + r$ ▷ end of span
            for $A \in N$ **do**
                for $m := i + 1$ **to** $j - 1$ **do** ▷ partition position
                    for $A \to BC \in R$ **do**
                        if $B \in c_{i,m}$ and $C \in c_{m,j}$ **then**
                            $c_{i,j} := c_{i,j} \cup \{A\}$

$c_{i,j} := \{A \mid i < m < j, A \to BC \in R\colon B \in c_{i,m}, C \in c_{m,j}\}$

    return $(c_{i,j} \mid 0 \le i < j \le n)$

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 $(c_{i,j} \colon N \to \mathbb{R} \mid 0 \leq i < j \leq n)$
update operation for rule $r = A \to \ldots$	$c_{i,j} := c_{i,j} \cup \{A\}$	$c_{i,j}(A) := \max\{c_{i,j}(A), p(r)\}$

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 $w = w_1 \ldots w_n$ where $w_1, \ldots, w_n \in \Sigma$
Ensure: family $(c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)$ such that

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

function CKY(R, p, $w_1 \ldots w_n$)
    $(c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N)$
    for $i := 1$ **to** n **do**
        for $A \to w_i \in R$ **do**
            $c_{i-1,i,A} := \max\{c_{i-1,i,A}, p(A \to w_i)\}$
    for $r := 2$ **to** n **do**
        for $i := 0$ **to** $n - r$ **do**
            $j := i + r$
            for $A \in N$ **do**
                for $m := i + 1$ **to** $j - 1$ **do**
                    for $A \to BC \in R$ **do**
                        $c_{i,j,A} := \max\{c_{i,j,A}, p(A \to BC) \cdot c_{i,m,B} \cdot c_{m,j,C}\}$
    return $(c_{i,j,A} \mid 0 \leq i < j \leq n, A \in N)$

Adding chain rules

Types of rules accepted by classic CKY: (with $A, B, C \in N$ and $w \in \Sigma$)

$$A \to w \qquad\qquad A \to BC$$

What we also want: $A \to B$ (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 $w = w_1 \ldots w_n$ where $w_1, \ldots, w_n \in \Sigma$
Ensure: family $(c_{i,j,A} \in \mathbb{R} \mid 0 \le i < j \le n, A \in N)$ such that

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

function CKY(R, μ, $w_1 \ldots w_n$)
  $(c_{i,j,A} := 0 \mid 0 \le i < j \le n, A \in N)$
  for $i := 1$ **to** n **do**
    for $A \to w_i \in R$ **do**
      $c_{i-1,i,A} := p(A \to w_i)$
    $(c_{i-1,i,A} \mid A \in N) = \text{UNARY\_CLOSURE}(R, p, (c_{i-1,i,A} \mid A \in N))$
  for $r := 2$ **to** n **do**
    for $i := 0$ **to** $n - r$ **do**
      $j := i + r$
      for $A \in N$ **do**
        for $m := i + 1$ **to** $j - 1$ **do**
          for $A \to BC \in R$ **do**
            $c_{i,j,A} := \max\{c_{i,j,A}, p(A \to BC) \cdot c_{i,m,B} \cdot c_{m,j,C}\}$
      $(c_{i,j,A} \mid A \in N) = \text{UNARY\_CLOSURE}(R, p, (c_{i,j,A} \mid A \in N))$
  return $(c_{i,j,A} \mid 0 \le i < j \le n, A \in N)$

The CKY parsing algorithm + weights + chain rules

function UNARY_CLOSURE($R, p, (c_A \mid A \in N)$)
    $queue := \{(A, c_A) \mid A \in N, c_A \neq 0\}$
    $(c_A := 0 \mid A \in N)$
    while $queue \neq \emptyset$ **do**
        $(B, q) := \operatorname{argmax}_{(\hat{B}, \hat{q}) \in queue} \hat{q}$
        $queue := queue \setminus \{(B, q)\}$
        if $c_B < q$ **then**
            $c_B := q$
            for $A \to B \in R$ **do**
                $queue := queue \cup \{(A, p(A \to B) \cdot q)\}$
    return $(c_A \mid A \in N)$

About backtraces

What we have: best weight $c_{i,j,A}$ for derivations in $D_G^A(w_{i+1}...w_j)$ for each $A \in N$, $0 \leq i < j \leq n$
What we want: the best derivation of S that results in $w = w_1 \ldots w_n$

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. $\bot$, or $A \to t$, or $(A \to B, i, j)$ or $(A \to BC, i, m, m, j)$

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. $A \to t$ **then return** $A \to t$
3: **else if** $b_{i,j,A}$ otf. $(A \to B, i, j)$ **then return** $(A \to B)$(BEST_TREE(b, i, j, B))
4: **else if** $b_{i,j,A}$ otf. $(A \to BC, i, m, m, j)$ **then**
5: **return** $(A \to BC)$(BEST_TREE(b, i, m, B), 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 `Map<Nt, (Rule, Wt)>`
  - for some, that be no concern `Set<(Rule, Wt)>`

Let's talk about data structures

- access to grammar rules depends on loops:
  - access by first nonterminal on rhs `Map<Nt, (Rule, Wt)>`
  - for some, that be no concern `Set<(Rule, Wt)>`
- weights for each span and nonterminal:
  - usually in a $(\frac{|w|\cdot(|w|+1)}{2} \cdot |N|)$-dimensional vector (dense) `Vec<Wt>`
  - or hashmap (sparse) `Map<(Int, Int, Nt), Wt>`

Let's talk about data structures

- access to grammar rules depends on loops:
  - access by first nonterminal on rhs `Map<Nt, (Rule, Wt)>`
  - for some, that be no concern `Set<(Rule, Wt)>`
- weights for each span and nonterminal:
  - usually in a $(\frac{|w| \cdot (|w|+1)}{2} \cdot |N|)$-dimensional vector (dense) `Vec<Wt>`
  - or hashmap (sparse) `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) `Vec<Bt>`
  - or in a hashmap (sparse) `Map<(Int, Int, Nt), Bt>`
  - or do not store them at all

Outline

Deduction systems [Ned03]

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

$$\frac{a_1 \qquad , \ldots, a_k}{c}$$

Deduction systems [Ned03]

- rule-based system (derivation of *items*)
- derive consequence (c) from antecedents ($a_1, \ldots, a_k$) for some $k \in \mathbb{N}$
- compute weight of consequence using weight of antecedents ($w_1, \ldots, w_k$)

$$\frac{a_1 : w_1, \ldots, a_k : w_k}{c\colon f(w_1,\ldots,w_k)}$$

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 ($w_1, \dots, w_k$)
- side condition b

$$\frac{a_1 : w_1, \dots, a_k : w_k}{c \colon f(w_1, \dots, w_k)}\ b$$

Deduction system for parsing weighted cfg [Ned03]

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

Deduction system for parsing weighted cfg [Ned03]

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

Deduction system for parsing weighted cfg [Ned03]

- item (i, A, j) for each nonterminal A spanning $w_{i+1} \ldots w_j$
- predict initial items $\frac{}{(i-1, A, i) \colon p(A \longrightarrow w_i)}$ $A \to w_i \in R \wedge w = w_1 \ldots w_i \ldots w_n$
- combine items $\frac{(i_0, B_1, i_1) \colon w_1, (i_1, B_2, i_2) \colon w_2, \ldots, (i_{k-1}, B_k, i_k) \colon w_k}{(i_0, A, i_k) \colon p(A \to B_1 \ldots B_k) \cdot w_1 \cdots w_k}$ $A \to B_1 \ldots B_k \in R$

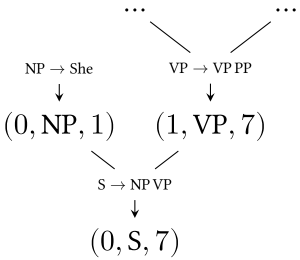
Deduction system for parsing weighted cfg [Ned03]

- item (i, A, j) for each nonterminal A spanning $w_{i+1} \ldots w_j$
- predict initial items $\frac{}{(i-1, A, i) : \, p(A \longrightarrow w_i)}$ $A \rightarrow w_i \in R \land w = w_1 \ldots w_i \ldots w_n$
- combine items $\frac{(i_0, B_1, i_1) : \, w_1, (i_1, B_2, i_2) : \, w_2, \ldots, (i_{k-1}, B_k, i_k) : \, w_k}{(i_0, A, i_k) : \, p(A \rightarrow B_1 \ldots B_k) \cdot w_1 \cdots w_k}$ $A \rightarrow B_1 \ldots B_k \in R$
- goal item: $(0, S, |w|)$

Deduction system for parsing weighted cfg [Ned03]

- item (i, A, j) for each nonterminal A spanning $w_{i+1} \ldots w_j$
- predict initial items $\frac{}{(i-1,A,i)\colon p(A \longrightarrow w_i)}\ A \to w_i \in R \wedge w = w_1 \ldots w_i \ldots w_n$
- combine items $\frac{(i_0,B_1,i_1)\colon w_1, (i_1,B_2,i_2)\colon w_2, \ldots, (i_{k-1},B_k,i_k)\colon w_k}{(i_0,A,i_k)\colon p(A \to B_1 \ldots B_k) \cdot w_1 \cdots w_k}\ A \to B_1 \ldots B_k \in R$
- 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 $w_1 \ldots w_n$ where $w_1, \ldots, w_n \in \Sigma$
Ensure: family $(c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)$ such that
$c_{i,j,A} = \max \left(\{p(d) \mid d \in D_G^A(w_i \ldots w_j)\} \cup \{0\} \right)$

```
function DEDUCE(R, p, w_1…w_n)
    queue := {(i − 1, A, i, p(A → w_i)) | 1 ≤ i ≤ n, A → w_i ∈ R}
    (c_{i,j,A} := 0 | 0 ≤ i < j ≤ n, A ∈ N)
    while queue ≠ ∅ do
        (i, A, j, q) := argmax_{(î, Â, ĵ, q̂) ∈ queue} q̂
        queue := queue \ {(i, A, j, q)}
        if c_{i,j,A} = 0 then
            c_{i,j,A} := q
            queue := queue ∪ {(i, A′, j′, p(A′ → AC) · q · c_{j,j′,C}) | j < j′ ≤ n, A′ → AC ∈ R}
            queue := queue ∪ {(i′, A′, j, p(A′ → BA) · c_{i′,i,B} · q) | 0 ≤ i′ < i, A′ → BA ∈ R}
            queue := queue ∪ {(i, A′, j, p(A′ → A) · q) | A′ → A ∈ R}
    return (c_{i,j,A} | 0 ≤ i < j ≤ n, A ∈ N)
```

Let's talk about data structures ... again

- access of grammar from each rhs nonterminal `Map<Nt, (Rule, Wt)>`

Let's talk about data structures … again

- access of grammar from each rhs nonterminal `Map<Nt, (Rule, Wt)>`
- each item may need to store a backtrace `(Int, Nt, Int, Wt, Bt)`

Let's talk about data structures … again

- access of grammar from each rhs nonterminal `Map<Nt, (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, (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

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

[1] Bodenstab [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

[1] Bodenstab [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

[1] Bodenstab [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
- try not to over-engineer it

[1] Bodenstab [Bod09] discusses this in detail.

[Bod09] N. Bodenstab. "Efficient Implementation of the cky algorithm". *Computational Linguistics, Final Project Paper,* 2009.

[CS70] J. 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] L. Huang and D. 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] M.-J. Nederhof. "Weighted deductive parsing and Knuth's algorithm". *Computational Linguistics*, 2003.

[You67] D. H. Younger. "Recognition and parsing of context-free languages in time n3". *Information and Control*, 1967.