Polynomial time parsing of PCFGs

Gerald Penn

(some slides from Pi-Chuan Chang and Christopher Manning)
0. Chomsky Normal Form

• All rules are of the form \( X \rightarrow Y Z \) or \( X \rightarrow w \).

• A transformation to this form doesn’t change the weak generative capacity of CFGs.
  • With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
  • Unaries/empties are removed recursively
  • \( n \)-ary rules introduce new nonterminals (\( n > 2 \))
    • \( VP \rightarrow V \ NP \ PP \) becomes \( VP \rightarrow V \ @VP-V \) and \( @VP-V \rightarrow NP \ PP \)

• In practice it’s a pain
  • Reconstructing \( n \)-aries is easy
  • Reconstructing unaries can be trickier

• But it makes parsing easier/more efficient
An example: before binarization…

cats scratch people with claws
After binarization...

cats scratch people with claws
Treebank: empties and unaries

PTB Tree | NoFuncTags | NoEmpties | High | Low
---------|------------|-----------|------|------
TOP      | S-HLN      | S         | S    | S    |
TOP      | NP-SUBJ    | NP        | VP   | VP   |
-NONE-   | VB         | -NONE-    | VB   | VB   |
ε        | Aton       | ε         | Aton | Aton |
Aton     | Aton       | Aton      | Aton |

-NONE-   | VB         | VB        | VB   |
ε        | Aton       | Aton      | Aton |
Aton     | Aton       | Aton      | Aton |

High     | NoUnaries  | Low       |
Constituency Parsing

PCFG

Rule Probs $\theta_i$

$\theta_0$: $S \rightarrow NP \ VP$

$\theta_1$: $NP \rightarrow NN \ NNS$

$\ldots$

$\theta_{42}$: $NN \rightarrow Factory$

$\theta_{43}$: $NNS \rightarrow payrolls$

$\ldots$

(factory payrolls fell in September)
Factory payrolls fell in September
Viterbi (Max) Scores

NP→NN NNS 0.13

\[ i_{NP} = (0.13)(0.0023)(0.0014) = 1.87 \times 10^{-7} \]

NP→NNP NNS 0.056

\[ i_{NP} = (0.056)(0.001)(0.0014) = 7.84 \times 10^{-8} \]
Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn’t increase algorithmic complexity
- Empty cases can be incorporated
  - Use fenceposts
  - Doesn’t increase complexity; essentially like unaries

- Binarization is *vital*
  - All sorts of optimizations depend on this
    - Binarization may be an explicit transformation or implicit in how the parser works (Early-style dotted rules), but it’s almost always there.
function CKY(words, grammar) returns most probable parse/prob
score = new double[#(words)+1][#(words)+1][#(nonterms)]
back = new Pair[#(words)+1][#(words)+1][#(nonterms)]
for i=0; i<#(words); i++
  for A in nonterms
    if A -> words[i] in grammar
      score[i][i+1][A] = P(A -> words[i])
//handle unaries
boolean added = true
while added
  added = false
  for A, B in nonterms
    if score[i][i+1][B] > 0 && A->B in grammar
      prob = P(A->B)*score[i][i+1][B]
      if(prob > score[i][i+1][A])
        score[i][i+1][A] = prob
        back[i][i+1] [A] = B
        added = true

The CKY algorithm (1960/1965) ... generalized
The CKY algorithm (1960/1965) ... generalized

for span = 2 to #(words)
    for begin = 0 to #(words)- span
        end = begin + span
        for split = begin+1 to end-1
            for A,B,C in nonterms
                prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
                if(prob > score[begin][end][A])
                    score[begin][end][A] = prob
                    back[begin][end][A] = new Triple(split,B,C)
        //handle unaries
        boolean added = true
        while added
            added = false
            for A, B in nonterms
                prob = P(A->B)*score[begin][end][B];
                if(prob > score[begin][end] [A])
                    score[begin][end] [A] = prob
                    back[begin][end] [A] = B
                    added = true

return buildTree(score, back)
<table>
<thead>
<tr>
<th></th>
<th>cats</th>
<th>1</th>
<th>scratch</th>
<th>2</th>
<th>walls</th>
<th>3</th>
<th>with</th>
<th>4</th>
<th>claws</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>score[0][1]</td>
<td></td>
<td></td>
<td></td>
<td>score[0][3]</td>
<td></td>
<td>score[0][5]</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>score[1][2]</td>
<td></td>
<td></td>
<td></td>
<td>score[1][4]</td>
<td></td>
<td>score[1][5]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>score[2][3]</td>
<td></td>
<td></td>
<td></td>
<td>score[2][5]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>score[3][4]</td>
<td></td>
<td></td>
<td></td>
<td>score[3][5]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>score[4][5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
for i=0; i<#(words); i++
    for A in nonterms
        if A -> words[i] in grammar
            score[i][i+1][A] = P(A -> words[i]);
<table>
<thead>
<tr>
<th></th>
<th>cats</th>
<th>scratch</th>
<th>walls</th>
<th>with</th>
<th>claws</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N→cats</td>
<td>P→cats</td>
<td>V→cats</td>
<td>NP→N</td>
<td>N→scratch</td>
</tr>
<tr>
<td>2</td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td>@VP→V→NP</td>
</tr>
<tr>
<td>4</td>
<td>N→claws</td>
<td>P→claws</td>
<td>V→claws</td>
<td>NP→N</td>
<td>N→claws</td>
</tr>
</tbody>
</table>

// handle unaries
<table>
<thead>
<tr>
<th>0</th>
<th>cats</th>
<th>1</th>
<th>scratch</th>
<th>2</th>
<th>walls</th>
<th>3</th>
<th>with</th>
<th>4</th>
<th>claws</th>
</tr>
</thead>
<tbody>
<tr>
<td>N→cats</td>
<td>P→cats</td>
<td>V→cats</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td>PP→P</td>
<td>@PP→P</td>
<td>VP→V</td>
<td>@VP→V</td>
</tr>
<tr>
<td>1</td>
<td>N→scratch</td>
<td>P→scratch</td>
<td>V→scratch</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td>PP→P</td>
<td>@PP→P</td>
<td>VP→V</td>
</tr>
<tr>
<td>2</td>
<td>N→walls</td>
<td>P→walls</td>
<td>V→walls</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td>PP→P</td>
<td>@PP→P</td>
<td>VP→V</td>
</tr>
<tr>
<td>3</td>
<td>N→with</td>
<td>P→with</td>
<td>V→with</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td>PP→P</td>
<td>@PP→P</td>
<td>VP→V</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>prob=score[begin][split][B]*score[split][end][C]*P(A→BC)</td>
<td>prob=score[0][1][P]*score[1][2][@PP→P] *P(PP→P @PP→P)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>N→claws</td>
<td>P→claws</td>
<td>V→claws</td>
<td>NP→N</td>
<td>@VP→V→NP</td>
<td>@PP→P→NP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For each A, only keep the “A→BC” with highest prob.
Call buildTree(score, back) to get the best parse.
Unary rules: alchemy in the land of treebanks
Same-Span Reachability

- ADJP
- ADVP
- FRAG
- INTJ
- NP
- PP
- PRN
- QP
- S
- SBAR
- UCP
- VP
- WHNP
- NX
- SQ
- X
- RRC
- SINV
- WHADJP
- SBARQ
- WHPP
- WHADVP
- LST
- CONJP
- NAC
- NoEmpties
Efficient CKY parsing

- CKY parsing can be made very fast (!), partly due to the simplicity of the structures used.
  - But that means a lot of the speed comes from engineering details
  - And a little from cleverer filtering
- Store chart as (ragged) 3 dimensional array of float (log probabilities)
  - score[start][end][category]
    - For treebank grammars the load is high enough that you don’t really gain from lists of things that were possible
    - 50 wds: (50x50)/2x(1000 to 20000)x4 bytes = 5–100MB for parse triangle. Large. (Can move to beam for span[i][j].)
- Use int to represent categories/words (Index)
Efficient CKY parsing

• Provide efficient grammar/lexicon accessors:
  • E.g., return list of rules with this left child category
  • Iterate over left child, check for zero (Neg. inf.) prob of X:[i,j] (abort loop), otherwise get rules with X on left

• Some X:[i,j] can be filtered based on the input string
  • Not enough space to complete a long flat rule?
  • No word in the string can be a CC?
    • Using a lexicon of possible POS for words gives a lot of constraint rather than allowing all POS for words
  • Cf. later discussion of figures-of-merit/A* heuristics
Which constituent (with probability) can you make?

- a) NP (2.3e-9)
- b) NP (4.6e-6)
- c) PP (.0004)
- d) VP (.0002)
3. Evaluating Parsing Accuracy

- Most sentences are not given a completely correct parse by any currently existing parser.
- For Penn Treebank parsing, the standard evaluation is over the number of correct constituents (labeled spans).
  - \([ \text{label}, \text{start}, \text{finish} ]\)
- A constituent is a triple, which must be exact in the true parse for the constituent to be marked correct.
- The LP/LR $F_1$ is the micro-averaged harmonic mean of labeled constituent precision and recall.
- This isn’t necessarily a great measure ... many people think dependency accuracy or raw data likelihood would be better.
Evaluation

(a) Brackets in gold standard tree (a.):
S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6:9), NP-(7,9), *NP-(9:10)

(b) Brackets in candidate parse:
S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:10), NP-(4:6), PP-(6:10), NP-(7,10)

(d) Precision: 3/8 = 37.5% Crossign Brackets: 0
Recall: 3/8 = 37.5% Crossing Accuracy: 100%
Labeled Precision: 3/8 = 37.5% Tagging Accuracy: 10/11 = 90.9%
Labeled Recall: 3/8 = 37.5%
How good are PCFGs?

- Robust (usually admit everything, but with low probability)
- Partial solution for grammar ambiguity: a PCFG gives some idea of the plausibility of a sentence
- But not so good because the independence assumptions are too strong
- Give a probabilistic language model
  - But in a simple case it performs worse than a trigram model
- WSJ parsing accuracy: about 73% LP/LR F1
- The problem seems to be that PCFGs lack the lexicalization of a trigram model
Putting words into PCFGs

- A PCFG uses the actual words only to determine the probability of parts-of-speech (the preterminals).
- In many cases we need to know about words to choose a parse.
- The head word of a phrase gives a good representation of the phrase’s structure and meaning.
  - Attachment ambiguities
    - *The astronomer saw the moon with the telescope*
  - Coordination
    - *the dogs in the house and the cats*
  - Subcategorization frames
    - *put versus like*
(Head) Lexicalization

- *put* takes both an NP and a VP
  - Sue put [ the book ]\textsubscript{NP} [ on the table ]\textsubscript{PP}
  - *Sue put [ the book ]\textsubscript{NP}
  - *Sue put [ on the table ]\textsubscript{PP}

- *like* usually takes an NP and not a PP
  - Sue likes [ the book ]\textsubscript{NP}
  - *Sue likes [ on the table ]\textsubscript{PP}

- We can’t tell this if we just have a VP with a verb, but we can if we know *what* verb it is
4. Accurate Unlexicalized Parsing: PCFGs and Independence

- The symbols in a PCFG define independence assumptions:

  \[ S \rightarrow NP \text{ VP} \]

  \[ NP \rightarrow DT \text{ NN} \]

- At any node, the material inside that node is independent of the material outside that node, given the label of that node.

- Any information that statistically connects behavior inside and outside a node must flow through that node.
Non-Independence I

- Independence assumptions are often too strong.

- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
Independence Assumptions

- PCFGs

- Lexicalized PCFGs

73% accuracy

88% accuracy

Michael Collins (2003, COLT)
Non-Independence II

- Who cares?
  - NB, HMMs, all make false assumptions!
  - For generation/LMs, consequences would be obvious.
  - For parsing, does it impact accuracy?

- Symptoms of overly strong assumptions:
  - Rewrites get used where they don’t belong.
  - Rewrites get used too often or too rarely.

In the PTB, this construction is for possessives
Breaking Up the Symbols

• We can relax independence assumptions by encoding dependencies into the PCFG symbols:

  Parent annotation
  [Johnson 98]

  Marking
  possessive NPs

• What are the most useful features to encode?
Annotations

• Annotations split the grammar categories into sub-categories.

• Conditioning on history vs. annotating
  • $P(\text{NP}^S \rightarrow \text{PRP})$ is a lot like $P(\text{NP} \rightarrow \text{PRP} | S)$
  • $P(\text{NP-POS} \rightarrow \text{NNP POS})$ isn’t history conditioning.

• Feature grammars vs. annotation
  • Can think of a symbol like $\text{NP}^{\text{NP-POS}}$ as $\text{NP} [\text{parent:NP}, +\text{POS}]$

• After parsing with an annotated grammar, the annotations are then stripped for evaluation.
Experimental Setup

- **Corpus**: Penn Treebank, WSJ

<table>
<thead>
<tr>
<th>Training:</th>
<th>Development:</th>
<th>Test:</th>
</tr>
</thead>
<tbody>
<tr>
<td>sections</td>
<td>section</td>
<td>section</td>
</tr>
<tr>
<td>02-21</td>
<td>22 (first 20 files)</td>
<td>23</td>
</tr>
</tbody>
</table>

- **Accuracy** – F1: harmonic mean of per-node labeled precision and recall.
- **Size** – number of symbols in grammar.
  - Passive / complete symbols: NP, NP^S
  - Active / incomplete symbols: NP → NP CC
Experimental Process

- We’ll take a highly conservative approach:
  - Annotate as sparingly as possible
  - Highest accuracy with fewest symbols
  - Error-driven, manual hill-climb, adding one annotation type at a time
Lexicalization

- Lexical heads are important for certain classes of ambiguities (e.g., PP attachment):

- Lexicalizing grammar creates a much larger grammar.
  - Sophisticated smoothing needed
  - Smarter parsing algorithms needed
  - More data needed

- How necessary is lexicalization?
  - Bilexical vs. monolexical selection
  - Closed vs. open class lexicalization
Unlexicalized PCFGs

- What do we mean by an “unlexicalized” PCFG?
  - Grammar rules are not systematically specified down to the level of lexical items
    - NP-stocks is not allowed
    - NP^S-CC is fine
  - Closed vs. open class words (NP^S-the)
    - Long tradition in linguistics of using function words as features or markers for selection
    - Contrary to the bilexical idea of semantic heads
    - Open-class selection really a proxy for semantics

- Honesty checks:
  - Number of symbols: keep the grammar very small
  - No smoothing: over-annotating is a real danger
Horizontal Markovization

- Horizontal Markovization: Merges States

Symbols

Horizontal Markov Order

Symbols

Horizontal Markov Order
Vertical Markovization

- Vertical Markov order: rewrites depend on past $k$ ancestor nodes. (cf. parent annotation)

Order 1

Order 2

Symbols

<table>
<thead>
<tr>
<th>Vertical Markov Order</th>
<th>1</th>
<th>2v</th>
<th>2</th>
<th>3v</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>72%</td>
<td>73%</td>
<td>74%</td>
<td>75%</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>73%</td>
<td>74%</td>
<td>75%</td>
<td>76%</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>74%</td>
<td>75%</td>
<td>76%</td>
<td>77%</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>75%</td>
<td>76%</td>
<td>77%</td>
<td>78%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertical Markov Order</th>
<th>1</th>
<th>2v</th>
<th>2</th>
<th>3v</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>72%</td>
<td>73%</td>
<td>74%</td>
<td>75%</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>73%</td>
<td>74%</td>
<td>75%</td>
<td>76%</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>74%</td>
<td>75%</td>
<td>76%</td>
<td>77%</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>75%</td>
<td>76%</td>
<td>77%</td>
<td>78%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbols</th>
<th>1</th>
<th>2v</th>
<th>2</th>
<th>3v</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>25600</td>
<td></td>
<td>12000</td>
<td>7000</td>
<td>4000</td>
<td>1000</td>
</tr>
<tr>
<td>13000</td>
<td></td>
<td>9000</td>
<td>5000</td>
<td>2000</td>
<td>1000</td>
</tr>
<tr>
<td>10500</td>
<td></td>
<td>7500</td>
<td>4500</td>
<td>2500</td>
<td>1000</td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td>7000</td>
<td>4000</td>
<td>2000</td>
<td>1000</td>
</tr>
<tr>
<td>9500</td>
<td></td>
<td>6500</td>
<td>3500</td>
<td>1500</td>
<td>1000</td>
</tr>
<tr>
<td>9000</td>
<td></td>
<td>6000</td>
<td>3000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>8500</td>
<td></td>
<td>5500</td>
<td>2500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>8000</td>
<td></td>
<td>5000</td>
<td>2000</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>7500</td>
<td></td>
<td>4500</td>
<td>1500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>7000</td>
<td></td>
<td>4000</td>
<td>1000</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>6500</td>
<td></td>
<td>3500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>6000</td>
<td></td>
<td>3000</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>5500</td>
<td></td>
<td>2500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td>2000</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>4500</td>
<td></td>
<td>1500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>4000</td>
<td></td>
<td>1000</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>3500</td>
<td></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>3000</td>
<td></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>2500</td>
<td></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>1500</td>
<td></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>1000</td>
</tr>
</tbody>
</table>
Examples:
- Raw treebank: \( v=1, h=\infty \)
- Johnson 98: \( v=2, h=\infty \)
- Collins 99: \( v=2, h=2 \)
- Best F1: \( v=3, h=2v \)

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base: ( v=h=2v )</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
</tbody>
</table>
Unary Splits

- Problem: unary rewrites used to transmute categories so a high-probability rule can be used.

- Solution: Mark unary rewrite sites with -U

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>UNARY</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
</tbody>
</table>
Tag Splits

- Problem: Treebank tags are too coarse.
- Example: Sentential, PP, and other prepositions are all marked IN.
- Partial Solution:
  - Subdivide the IN tag.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>80.3</td>
<td>8.1K</td>
</tr>
</tbody>
</table>
Other Tag Splits

- **UNARY-DT**: mark demonstratives as DT^U ("the X" vs. "those")
- **UNARY-RB**: mark phrasal adverbs as RB^U ("quickly" vs. "very")
- **TAG-PA**: mark tags with non-canonical parents ("not" is an RB^VP)
- **SPLIT-AUX**: mark auxiliary verbs with –AUX [cf. Charniak 97]
- **SPLIT-CC**: separate “but” and “&” from other conjunctions
- **SPLIT-%**: “%” gets its own tag.

<table>
<thead>
<tr>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.4</td>
<td>8.1K</td>
</tr>
<tr>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>81.2</td>
<td>8.5K</td>
</tr>
<tr>
<td>81.6</td>
<td>9.0K</td>
</tr>
<tr>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td>81.8</td>
<td>9.3K</td>
</tr>
</tbody>
</table>
Yield Splits

- Problem: sometimes the behavior of a category depends on something inside its future yield.

- Examples:
  - Possessive NPs
  - Finite vs. infinite VPs
  - Lexical heads!

- Solution: annotate future elements into nodes.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>82.3</td>
<td>9.7K</td>
</tr>
<tr>
<td>POSS-NP</td>
<td>83.1</td>
<td>9.8K</td>
</tr>
<tr>
<td>SPLIT-VP</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
</tbody>
</table>
Distance / Recursion Splits

- Problem: vanilla PCFGs cannot distinguish attachment heights.

- Solution: mark a property of higher or lower sites:
  - Contains a verb.
  - Is (non)-recursive.
    - Base NPs [cf. Collins 99]
    - Right-recursive NPs

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
<tr>
<td>BASE-NP</td>
<td>86.0</td>
<td>11.7K</td>
</tr>
<tr>
<td>DOMINATES-V</td>
<td>86.9</td>
<td>14.1K</td>
</tr>
<tr>
<td>RIGHT-REC-NP</td>
<td>87.0</td>
<td>15.2K</td>
</tr>
</tbody>
</table>
A Fully Annotated Tree
Final Test Set Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>LP</th>
<th>LR</th>
<th>F1</th>
<th>CB</th>
<th>0 CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magerman 95</td>
<td>84.9</td>
<td>84.6</td>
<td>84.7</td>
<td>1.26</td>
<td>56.6</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td>86.0</td>
<td>1.14</td>
<td>59.9</td>
</tr>
<tr>
<td>Klein &amp; M 03</td>
<td>86.9</td>
<td>85.7</td>
<td>86.3</td>
<td>1.10</td>
<td>60.3</td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td>87.4</td>
<td>1.00</td>
<td>62.1</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
<td>0.90</td>
<td>67.1</td>
</tr>
</tbody>
</table>

- Beats “first generation” lexicalized parsers.