EVALB, Improving CKY Parsing, Hw3

Scott Farrar
CLMA, University of Washington
farrar@u.washington.edu

January 28, 2010
Today’s lecture

1. Evaluating parsers

2. Hw3

3. Optimization: tips and tricks
   1. Size of the grammar
   2. Limit rules added to chart
   3. Sentence length
The Wall Street Journal (WSJ) section of the Penn Treebank (PTB), for all its faults, provides a very useful resource for comparing parser performance.
The Wall Street Journal (WSJ) section of the Penn Treebank (PTB), for all its faults, provides a very useful resource for comparing parser performance.

In building a probabilistic parser, there are four kinds of resources that are commonly used esp. in the ACL related literature:

1. **Size of the grammar**
2. **Limit rules added to chart**
3. **Sentence length**

---

**Optimization: tips and tricks**

- 1. Size of the grammar
- 2. Limit rules added to chart
- 3. Sentence length
Parsing: \textit{dev/train/test} paradigm

The Wall Street Journal (WSJ) section of the Penn Treebank (PTB), for all its faults, provides a very useful resource for comparing parser performance.

In building a probabilistic parser, there are four kinds of resources that are commonly used esp. in the ACL related literature:

1. **training data**: large number of annotated sentences (sec. 2–21 of PTB has 39,830 sentences)
Parsing: dev/train/test paradigm

The Wall Street Journal (WSJ) section of the Penn Treebank (PTB), for all its faults, provides a very useful resource for comparing parser performance.

In building a probabilistic parser, there are four kinds of resources that are commonly used esp. in the ACL related literature:

1. **training data**: large number of annotated sentences (sec. 2–21 of PTB has 39,830 sentences)

2. **development data**: small number of annotated sentences used to "tweak" parser (sec. 22, of PTB)
The Wall Street Journal (WSJ) section of the Penn Treebank (PTB), for all its faults, provides a very useful resource for comparing parser performance.

In building a probabilistic parser, there are four kinds of resources that are commonly used esp. in the ACL related literature:

1. **training data**: large number of annotated sentences (sec. 2–21 of PTB has 39,830 sentences)
2. **development data**: small number of annotated sentences used to “tweak” parser (sec. 22, of PTB)
3. **test data**: small-medium number of un-annotated sentences used as input to parser (sec. 23 of PTB has 2416 sentences, ∼ 6% of training set)
Parsing: dev/train/test paradigm

The Wall Street Journal (WSJ) section of the Penn Treebank (PTB), for all its faults, provides a very useful resource for comparing parser performance.

In building a probabilistic parser, there are four kinds of resources that are commonly used esp. in the ACL related literature:

1. **training data**: large number of annotated sentences (sec. 2–21 of PTB has 39,830 sentences)
2. **development data**: small number of annotated sentences used to “tweak” parser (sec. 22, of PTB)
3. **test data**: small-medium number of un-annotated sentences used as input to parser (sec. 23 of PTB has 2416 sentences, ~ 6% of training set)
4. **gold standard**: annotated version of test data, with no errors (hidden till parser is developed)
Recall our discussion first day of class

**Definition**

**objective criterion**: that which a parser tries to maximize.
Recall our discussion first day of class

**Definition**

**objective criterion**: that which a parser tries to maximize.

**Definition**

**tree accuracy**: (harsh) exact match criterion; 1 for perfect match, otherwise 0.
Recall our discussion first day of class

Definition

**objective criterion**: that which a parser tries to maximize.

Definition

**tree accuracy**: (harsh) exact match criterion; 1 for perfect match, otherwise 0.

Non-exact matches can be very useful for some tasks: named entity extraction, information retrieval, document clustering.
PARSEVAL

**Definition**

**PARSEVAL measures**: standard metrics for evaluation using the component pieces of a parse; a way to give partial credit.

evalb is an implementation of the PARSEVAL measures.

The evalb program uses several PARSEVAL measures:

- labeled precision (LP)
- labeled recall (LR)
- F-measure
- cross bracketing
PARSEVAL: Labeled precision

Definition

Labeled Precision (LP): the average of how many brackets in the resulting parse tree match those in the gold standard (same span). Focusing in on specific problems can increase precision. Broadening your methodology can decrease precision. Labeled precision includes the node label as well.

\[ LP = \frac{\text{# of correct constituents in candidate parse of } s}{\text{# of total constituents in candidate parse of } s} \]
PARSEVAL: Labeled recall

Definition

Labeled Recall (LR): the average of how many brackets in the gold standard are in the resulting parse. Did you get them all? Coverage. Focusing in on specific problems can decrease recall, because other problems may get ignored. Labeled recall includes the node label as well.

\[
LR = \frac{\text{# of correct constituents in candidate parse of } s}{\text{# of correct constituents in reference parse of } s}
\]
P, R errors

Example

PP attachment error

(S (NP (A a)) (VP(B b) (PP (C c))))) gold
(S (NP (A a)) (VP(B b)) (PP (C c)))

Precision $P = \frac{3}{4}$
Recall $R = \frac{3}{4}$

8/42
P, R errors

Example

PP attachment error

(S (NP (A a)) (VP(B b) (PP (C c)))) ) gold
(S (NP (A a)) (VP(B b) ) (PP (C c)))

Constituents in gold: $S(0, 3), NP(0, 1), VP(1, 3), PP(2, 3)$
P, R errors

Example

PP attachment error

(S (NP (A a)) (VP(B b) (PP (C c))) ) gold
(S (NP (A a)) (VP(B b) ) (PP (C c)))

Constituents in gold: $S(0, 3), \ NP(0, 1), \ VP(1, 3), \ PP(2, 3)$
Constituents in cand: $S(0, 3), \ NP(0, 1), \ VP(1, 2), \ PP(2, 3)$
P, R errors

Example

PP attachment error

(S (NP (A a)) (VP(B b) (PP (C c))))  gold
(S (NP (A a)) (VP(B b)) (PP (C c)))

Constituents in gold:  S(0, 3), NP(0, 1), VP(1, 3), PP(2, 3)
Constituents in cand:  S(0, 3), NP(0, 1), VP(1, 2), PP(2, 3)

Precision

\[ P = \frac{3}{4} \]
P, R errors

Example

PP attachment error

(S (NP (A a)) (VP(B b) (PP (C c))))) gold
(S (NP (A a)) (VP(B b)) (PP (C c)))

Constituents in gold: S(0, 3), NP(0, 1), VP(1, 3), PP(2, 3)
Constituents in cand: S(0, 3), NP(0, 1), VP(1, 2), PP(2, 3)

Precision

\[ P = \frac{3}{4} \]

Recall

\[ P = \frac{3}{4} \]
PARSEVAL: F-measure

Definition

F-measure is the weighted aggregation of precision and recall (harmonic mean).
**PARSEVAL: F-measure**

**Definition**

*F-measure* is the weighted aggregation of precision and recall (harmonic mean).

\[
F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}
\]

0 ≤ β ≤ +∞

- When β is 1, P and R are weighted equally.
- When β is greater than 1, R is favored.
- When β is less than 1, P is favored.
PARSEVAL: F-measure

Equally weighted $P$ and $R$

$$F_1 = \frac{(1^2 + 1) \times 0.9 \times 0.3}{1^2 \times 0.9 + 0.3} = 0.45$$
EVALB: Improving CKY Parsing, Hw3
Scott Farrar
CLMA, University of Washington
farrar@u.washington.edu

Evaluating parsers
Hw3
Optimization: tips and tricks
1. Size of the grammar
2. Limit rules added to chart
3. Sentence length

PARSEVAL: F-measure

Equally weighted $P$ and $R$

$$F_1 = \frac{\left(1^2 + 1\right) \times 0.9 \times 0.3}{1^2 \times 0.9 + 0.3} = 0.45$$

Harmonic Mean

$F_1$ is the same as the harmonic mean:

$$HM(a_1, a_2, a_3, \ldots, a_n) = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \frac{1}{a_3} + \ldots + \frac{1}{a_n}}$$

$$\frac{2}{\frac{1}{0.9} + \frac{1}{0.3}} = 0.45$$
EVALB, Improving CKY Parsing, Hw3

Scott Farrar
CLMA, University of Washington farrar@u.washington.edu

Evaluating parsers
Hw3
Optimization: tips and tricks
1. Size of the grammar
2. Limit rules added to chart
3. Sentence length

PARSEVAL: F-measure

Favoring $R$

\[ F_2 = \frac{(2^2 + 1) \times 0.9 \times 0.3}{2^2 \times 0.9 + 0.3} = 0.346 \]

Favoring $P$

\[ F_{.5} = \frac{(0.5^2 + 1) \times 0.9 \times 0.3}{0.5^2 \times 0.9 + 0.3} = 0.643 \]

What is $F_0$?

\[ F_0 = \frac{2^2 + 1}{2^2 \times 0.9 + 0.3} = 0.9 = \frac{11}{12} \]
EVALB, Improving CKY Parsing, Hw3

Scott Farrar
CLMA, University of Washington farrar@u.washington.edu

Evaluating parsers

Hw3

Optimization: tips and tricks
1. Size of the grammar
2. Limit rules added to chart
3. Sentence length

PARSEVAL: F-measure

Favoring \( R \)

\[
F_2 = \frac{(2^2 + 1) \times 0.9 \times 0.3}{2^2 \times 0.9 + 0.3} = 0.346
\]

Favoring \( P \)

\[
F_{5.5} = \frac{(.5^2 + 1) \times 0.9 \times 0.3}{.5^2 \times 0.9 + 0.3} = 0.643
\]

What is \( F_0 \)?

\[
F_0 = \frac{(0^2 + 1) \times 0.9 \times 0.3}{0^2 \times 0.9 + 0.3} = 0.9 = P
\]
PARSEVAL: cross-bracketing

**Definition**

**cross-bracketing**: the average of how many constituents in the resulting parse tree cross over the brackets in the gold standard.

**Example**

Candidate

```
( ( ( ( ) ) ) )
```

Gold std

```
( ( ( ( ) ) ) )
```

```
w1  w2  w3  w4  w5  w6  w7  w8
```

One cross-bracket error.
PARSEVAL: cross-bracketing

Example

Candidate

( ( ( ) ) )

Gold std

( ( ( ) ) )

w1 w2 w3 w4 w5 w6 w7 w8

Also one cross-bracket error.
PARSEVAL: Perfect results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>37</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>29</td>
<td>29</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>...........</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2413</td>
<td>23</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>2414</td>
<td>37</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>2415</td>
<td>15</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>2416</td>
<td>13</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100.00</td>
<td>100.00</td>
<td>49749</td>
<td>49749</td>
</tr>
</tbody>
</table>
PARSEVAL: Perfect results

=== Summary ===

-- All --
Number of sentence = 2416
Number of Error sentence = 0
Number of Skip sentence = 0
Number of Valid sentence = 2416
Bracketing Recall = 100.00
Bracketing Precision = 100.00
Complete match = 100.00
Average crossing = 0.00
No crossing = 100.00
2 or less crossing = 100.00
Tagging accuracy = 100.00
PARSEVAL: Perfect results

-- len<=40 --
Number of sentence = 2160
Number of Error sentence = 0
Number of Skip sentence = 0
Number of Valid sentence = 2160
Bracketing Recall = 100.00
Bracketing Precision = 100.00
Complete match = 100.00
Average crossing = 0.00
No crossing = 100.00
2 or less crossing = 100.00
Tagging accuracy = 100.00
No. of matched brackets = 39896
No. of gold brackets = 39896
No. of test brackets = 39896

Optimization: tips and tricks
1. Size of the grammar
2. Limit rules added to chart
3. Sentence length
P, R errors

Example

Cross bracket error

(S (NP (A a) (B b) ) (VP(C c) (PP (D d))))) gold
(S (NP (A a) ) (VP (B b) (C c) (PP (D d)))))

Precision $P = \frac{2}{4} = 0.5$
Recall $R = \frac{2}{4} = 0.5$

1 cross-bracket error
17/42
P, R errors

Example

Cross bracket error

(S (NP (A a) (B b)) (VP (C c) (PP (D d)))) gold
(S (NP (A a)) (VP (B b) (C c) (PP (D d))))

Constituents (gold): S(0, 4), NP(0, 2), VP(2, 4), PP(3, 4)
Constituents (cand): S(0, 4), NP(0, 1), VP(1, 4), PP(3, 4)
P, R errors

Example

Cross bracket error

(S (NP (A a) (B b)) (VP (C c) (PP (D d))))
(S (NP (A a)) (VP (B b) (C c) (PP (D d))))

Constituents (gold): S(0, 4), NP(0, 2), VP(2, 4), PP(3, 4)
Constituents (cand): S(0, 4), NP(0, 1), VP(1, 4), PP(3, 4)

Precision

$P = \frac{2}{4}$
P, R errors

Example

Cross bracket error

\[(S (NP (A a) (B b) ) (VP(C c) (PP (D d))))]\] gold
\[(S (NP (A a ) ) (VP (B b) (C c) (PP (D d))))]\]

Constituents (gold): \(S(0, 4), NP(0, 2), VP(2, 4), PP(3, 4)\)
Constituents (cand): \(S(0, 4), NP(0, 1), VP(1, 4), PP(3, 4)\)

Precision

\[P = \frac{2}{4}\]

Recall

\[P = \frac{2}{4}\]

Cross-bracket

1 cross-bracket error
Explanation of PARSEVAL

Have a look at the parameters files in dropbox/.../571/tools/EVALB

See Manning & Schutze (1999), p. 433
Today’s lecture

1. Evaluating parsers

2. Hw3

3. Optimization: tips and tricks
   - 1. Size of the grammar
   - 2. Limit rules added to chart
   - 3. Sentence length
Homework 3

See website

Optimization: tips and tricks
1. Size of the grammar
2. Limit rules added to chart
3. Sentence length

CNF grammar
There's no need to use your 2CNF code, but knowing how the grammar was transformed is important.

Unary rules:
S, VP, NP, etc.

Non-binary rules:
VP', NP', etc.
Homework 3

See website

CNF grammar

There’s no need to use your 2CNF code, but knowing how the grammar was transformed is important.

- Unary rules: \( S \rightarrow VP, NP \rightarrow NP \), etc.
- Non-binary rules: \( VP' \), \( NP' \), etc.
Collapsed Unaries

\[ S \_ VP \rightarrow VB \_ NP \]

was originally:
Collapsed Unaries

\[ S \_ VP \rightarrow VB \_ NP \]

was originally:
\[ S \rightarrow VP \]
\[ VP \rightarrow VB \_ NP \]
Hw3 Grammar

Binarized Productions

\[ VP \rightarrow VP' \ PP \]

where

\[ VP' \rightarrow VB \ PP \]

was originally:

\[ VP \rightarrow VP' \ PP \ PP \]
Hw3 Grammar

Binarized Productions

\[ VP \rightarrow VP' PP \]

where

\[ VP' \rightarrow VB PP \]

was originally:

\[ VP \rightarrow VB PP PP \]
Hw3 Grammar

Combination

\[ S \rightarrow VP \]

was originally:

```
S  
|   
VP  
|   
  |   
VB NP NP  
| |   
Show PRP ...  
| |   
  | me
```
Hw3 Grammar

```
S_VP
|   |   |
VB  NP_PRP  NP
|   |   | ...
Show me ...
```

Evaluating parsers

**Optimization: tips and tricks**

1. Size of the grammar
2. Limit rules added to chart
3. Sentence length
Hw3 Grammar

Evaluating parsers

**Optimization: tips and tricks**

1. Size of the grammar
2. Limit rules added to chart
3. Sentence length
Amendment to Task 4

Accuracy
You’re asked to improve upon the baseline parser so that you get a better EVALB score.
Amendment to Task 4

Accuracy
You’re asked to improve upon the baseline parser so that you get a better EVALB score.

Efficiency
We’ll also accept improved parsers that are more efficient, not necessarily more accurate. That is, improve the runtime of the parser without significantly degrading the efficiency.
Today’s lecture

1. Evaluating parsers

2. Hw3

3. Optimization: tips and tricks
   - 1. Size of the grammar
   - 2. Limit rules added to chart
   - 3. Sentence length
General strategies

The efficiency of the basic CYK is $O(n^3|P|)$, where $n$ is the average length of sentence and $|P|$ is the number of production rules. You can improve the efficiency by:

1. Limiting the size of the grammar
2. Limiting the number of states entered into the CKY chart (prune search space)
3. Reducing $n$, where $n$ is the length of the input sentence

Trade-off

There is always a speed vs. accuracy trade-off in statistical parsing. Where’s the sweet spot?
General strategies

The efficiency of the basic CYK is $O(n^3|P|)$, where $n$ is the average length of sentence and $|P|$ is the number of production rules. You can improve the efficiency by:

1. Limiting the size of the grammar $|P|$.
2. Limiting the number of states entered into the CKY chart (prune search space).
3. Reducing $n$, where $n$ is the length of the input sentence.
General strategies

The efficiency of the basic CYK is $O(n^3|P|)$, where $n$ is the average length of sentence and $|P|$ is the number of production rules. You can improve the efficiency by:

1. limiting the size of the grammar $|P|$
General strategies

The efficiency of the basic CYK is $O(n^3|P|)$, where $n$ is the average length of sentence and $|P|$ is the number of production rules. You can improve the efficiency by:

1. limiting the size of the grammar $|P|$
2. limiting the number of states entered into the CKY chart (prune search space)
General strategies

The efficiency of the basic CYK is $O(n^3|P|)$, where $n$ is the average length of sentence and $|P|$ is the number of production rules. You can improve the efficiency by:

1. limiting the size of the grammar $|P|$  
2. limiting the number of states entered into the CKY chart (prune search space)  
3. reducing $n$, where $n$ is the length of the input sentence

Trade-off

There is always a speed vs. accuracy trade-off in statistical parsing. Where’s the sweet spot?
General strategies

The efficiency of the basic CYK is $O(n^3|P|)$, where $n$ is the average length of sentence and $|P|$ is the number of production rules. You can improve the efficiency by:

1. Limiting the size of the grammar $|P|$
2. Limiting the number of states entered into the CKY chart (prune search space)
3. Reducing $n$, where $n$ is the length of the input sentence

Trade-off

There is always a speed vs. accuracy trade-off in statistical parsing. Where’s the sweet spot?
Limit the size of the grammar

In a wide-coverage grammar, you will have 1,000s of rule types (CYK requires you to search the rule store over and over). To handle a large number of rules, avoid creating so many rules to begin with.

Conversion to CNF is the major cause of rule proliferation. We can also prune away less important rules using a number of other techniques (recall Giazauskus paper).

Optimization: tips and tricks
1. Size of the grammar
2. Limit rules added to chart
3. Sentence length
Limit the size of the grammar

- In a wide-coverage grammar, you will have 1,000s of rule types (CYK requires you to search the rule store over and over).

In a wide-coverage grammar, you will have 1,000s of rule types (CYK requires you to search the rule store over and over).
Limit the size of the grammar

- In a wide-coverage grammar, you will have 1,000s of rule types (CYK requires you to search the rule store over and over).
- To handle a large number of rules, avoid creating so many rules to begin with.
Limit the size of the grammar

- In a wide-coverage grammar, you will have 1,000s of rule types (CYK requires you to search the rule store over and over).
- To handle a large number of rules, avoid creating so many rules to begin with.
- Conversion to CNF is the major cause of rule proliferation.
Limit the size of the grammar

- In a wide-coverage grammar, you will have 1,000s of rule types (CYK requires you to search the rule store over and over).
- To handle a large number of rules, avoid creating so many rules to begin with.
- Conversion to CNF is the major cause of rule proliferation.
- We can also prune away less important rules using a number of other techniques (recall Giazauskus paper).
Binarization choices

Original tree in grammar

```
A
/ |  \
B C D
```
Binarization choices

Original tree in grammar

```
A
/  \
B  C  D
```

Right-factored

```
A
/  \
B  X1
/  \
C  D
```
Binarization choices

Original tree in grammar

```
A
/ | \ 
B C D
```

Optimization: tips and tricks
1. Size of the grammar
2. Limit rules added to chart
3. Sentence length

Be sure to see write-up of CNF conversion in the NLTK documentation of nltk.treetransforms.
Binarization choices

Original tree in grammar

```
A
/  \ /  \B  C  D
```

Left-factored

```
A
/  \ /  
X1 D
/  \B  C
```

Be sure to see write-up of CNF conversion in the NLTK documentation of `nltk.treetransforms`. 
Parent Rule Annotation

**Definition**

*Parent rule annotation* refers to the annotation of nodes with information about their ancestor nodes, as if you're giving the nodes a context. Could improve CKY from 74% to 79% accuracy.

**Example**

<table>
<thead>
<tr>
<th>Original</th>
<th>Parent Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A^&lt;P&gt;</td>
</tr>
<tr>
<td>/ \</td>
<td>/ \</td>
</tr>
<tr>
<td>B C D ==&gt;</td>
<td>B^&lt;A&gt; X1^&lt;P&gt;</td>
</tr>
<tr>
<td></td>
<td>/ \</td>
</tr>
<tr>
<td></td>
<td>C^&lt;A&gt; D^&lt;A&gt;</td>
</tr>
</tbody>
</table>
Horizontal factoring

**Definition**

**Horizontal factoring** refers to the way in which rules in the original grammar can be binarized such that information about the child nodes is encoded in new nodes (in CNF). Also called Markovization, this captures “context” among terminals. As the Markov order increases, the number rules in the converted CFG increases, but more information is captured in rules. Data sparsity is, as usual, a big problem.

**Example**

Original Markov order 0

```
  ___A___
 /  /\  \
B  C  D  E  F  ==>  B  X1
 /  /\  \
C  __X2__
 /  /\  \
D    ....
```

Horizontal factoring defines the process of converting a grammar into a horizontal factored form, where each rule is binarized and the information about child nodes is encoded in new nodes. This is especially useful in the context of parser optimization, as it allows for more efficient parsing by reducing the number of rules and capturing more context in the grammar. However, it also introduces data sparsity issues, which can be a big problem.
Horizontal factoring

Example

Markov order 1  Markov order 2  etc.

A
/ \  / \  
B  A|<C>  ==>  B  A|<C-D>
/ \  / \  
C  ...  C  ...

Size of the grammar
Limit rules added to chart
Sentence length
Horizontal factoring

Example

Original: No smoothing, or order infinity

```
__A__   A
/    /     /    \
B  C  D  E  F ==> B  A|<C-D-E-F>
     /    \
     C    ...
```

Optimization: tips and tricks
1. Size of the grammar
2. Limit rules added to chart
3. Sentence length
Affects Markov order-N smoothing on rule size

As reported in Mohri and Roark (2006)

Sections 02-23 of PTB-WSJ:

- Markov factor 0: 99 nonterminals, 3803 productions
- Markov factor 1: 564 nonterminals, 6354 productions
- Markov factor 2: 2492 nonterminals, 11659 productions
- Markov factor $\infty$: 10,105 nonterminals, 23,220 productions
EVALB, Improving CKY Parsing, Hw3

Scott Farrar
CLMA, University of Washington
farar@u.washington.edu

Evaluating parsers

Hw3

Optimization: tips and tricks

1. Size of the grammar
2. Limit rules added to chart
3. Sentence length

Affects Markov order-N smoothing on rule size

As reported in Mohri and Roark (2006)

Sections 02-23 of PTB-WSJ:

- Markov factor 0: 99 nonterminals, 3803 productions
Affects Markov order-N smoothing on rule size

As reported in Mohri and Roark (2006)

Sections 02-23 of PTB-WSJ:

- Markov factor 0: 99 nonterminals, 3803 productions
- Markov factor 1: 564 nonterminals, 6354 productions

As reported in Mohri and Roark (2006)
Affects Markov order-N smoothing on rule size

As reported in Mohri and Roark (2006)

Sections 02-23 of PTB-WSJ:

- Markov factor 0: 99 nonterminals, 3803 productions
- Markov factor 1: 564 nonterminals, 6354 productions
- Markov factor 2: 2492 nonterminals, 11659 productions

As reported in Mohri and Roark (2006)
Affects Markov order-N smoothing on rule size

As reported in Mohri and Roark (2006)

Sections 02-23 of PTB-WSJ:

- Markov factor 0: 99 nonterminals, 3803 productions
- Markov factor 1: 564 nonterminals, 6354 productions
- Markov factor 2: 2492 nonterminals, 11659 productions
- Markov factor ∞: 10,105 nonterminals, 23,220 productions
## Combined Effects of Markov ordering and Parent annotation

<table>
<thead>
<tr>
<th>PCFG</th>
<th>Time(s)</th>
<th>Words/s</th>
<th>NTs</th>
<th>Prods</th>
<th>LR</th>
<th>LP</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right-factored, M-∞</td>
<td>4848</td>
<td>6.7</td>
<td>10105</td>
<td>23220</td>
<td>69.2</td>
<td>73.8</td>
<td>71.5</td>
</tr>
<tr>
<td>Right-factored, M–2</td>
<td>1302</td>
<td>24.9</td>
<td>2492</td>
<td>11659</td>
<td>68.8</td>
<td>73.8</td>
<td>71.3</td>
</tr>
<tr>
<td>Right-factored, M–1</td>
<td>445</td>
<td>72.7</td>
<td>564</td>
<td>6354</td>
<td>68.0</td>
<td>73.0</td>
<td>70.5</td>
</tr>
<tr>
<td>Right-factored, M–0</td>
<td>206</td>
<td>157.1</td>
<td>99</td>
<td>3803</td>
<td>61.5</td>
<td>65.5</td>
<td>63.3</td>
</tr>
<tr>
<td>Parent-annot., Rt-f M-2</td>
<td>7510</td>
<td>4.3</td>
<td>5876</td>
<td>22444</td>
<td>76.2</td>
<td>78.3</td>
<td>77.2</td>
</tr>
</tbody>
</table>
Converting to CNF

Note: different notation used for horizontal annotations.

In general, using no (or $\infty$ horizontal) factoring will give you better accuracy. But we have a rule explosion problem, so we’ll compromise our accuracy for better parser runtime. (See Mohri & Roark 2006).
2. Limit rules entered into chart

**Definition**

Use **beam thresholding**, named after the evaluation function in a beam search algorithm. Beam search is a way to only explore nodes in a search tree that are most likely to yield an answer. Some strategies:
2. Limit rules entered into chart

Definition

Use **beam threshholding**, named after the evaluation function in a beam search algorithm. Beam search is a way to only explore nodes in a search tree that are most likely to yield an answer. Some strategies:

- **Size of the grammar**
- **Limit rules added to chart**
- **Sentence length**

- **Problem?**
  You aren't guaranteed to always find an answer (a parse).
2. Limit rules entered into chart

Definition

Use **beam thresholding**, named after the evaluation function in a beam search algorithm. Beam search is a way to only explore nodes in a search tree that are most likely to yield an answer. Some strategies:

- Using beam width $k$ (only allow $k$ entries in cell):
  - $k=10$, 100, or 200
2. Limit rules entered into chart

**Definition**

Use **beam thresholding**, named after the evaluation function in a beam search algorithm. Beam search is a way to only explore nodes in a search tree that are most likely to yield an answer. Some strategies:

- Using beam width $k$ (only allow $k$ entries in cell):
  - $k=10, 100, \text{ or } 200$
- With your small grammar: $k=2, 5, \text{ or } 10$
2. Limit rules entered into chart

Definition

Use **beam thresholding**, named after the evaluation function in a beam search algorithm. Beam search is a way to only explore nodes in a search tree that are most likely to yield an answer. Some strategies:

- Using beam width $k$ (only allow $k$ entries in cell): $k = 10, 100, \text{or } 200$
- With your small grammar: $k = 2, 5, \text{or } 10$
- Remove all production rules with a frequency of 1 from grammar. Then try 2 and possibly 3. (variant)
2. Limit rules entered into chart

Definition

Use **beam thresholding**, named after the evaluation function in a beam search algorithm. Beam search is a way to only explore nodes in a search tree that are most likely to yield an answer. Some strategies:

- Using beam width $k$ (only allow $k$ entries in cell): $k=10$, 100, or 200
- With your small grammar: $k=2$, 5, or 10
- Remove all production rules with a frequency of 1 from grammar. Then try 2 and possibly 3. (variant)

Problem?

You aren’t guaranteed to always find an answer (a parse).
2. Limit rules entered into chart

**Definition**

Use **beam thresholding**, named after the evaluation function in a beam search algorithm. Beam search is a way to only explore nodes in a search tree that are most likely to yield an answer. Some strategies:

- Using beam width $k$ (only allow $k$ entries in cell): $k=10, 100, \text{ or } 200$
- With your small grammar: $k=2, 5, \text{ or } 10$
- Remove all production rules with a frequency of 1 from grammar. Then try 2 and possibly 3. (variant)

**Problem?**

You aren’t guaranteed to always find an answer (a parse).
More strategies

Heuristics

Within the CYK algorithm, heuristically throw away constituents that probably won’t make it into a complete parse. In other words limit the number of nodes saved in each cell of the CYK table.
More strategies

Heuristics
Within the CYK algorithm, heuristically throw away constituents that probably won’t make it into a complete parse. In other words limit the number of nodes saved in each cell of the CYK table.

Where $x$ and $y$ are constituents:

- Throw constituent $x$ away if $p(x) < 10^{-20}$.
- Throw $x$ away if $p(x) < 100 \times p(y)$ for some $y$ that spans the same set of words.
- Throw away $NP_{i,j}$ b/c $p(NP_{i,j}) = 0$.
- $p(VP_{i,j}) = 0$.
More strategies

Heuristics

Within the CYK algorithm, heuristically throw away constituents that probably won’t make it into a complete parse. In other words limit the number of nodes saved in each cell of the CYK table.

Where $x$ and $y$ are constituents:

- Throw constituent $x$ away if $p(x) < 10^{-200}$. 

- Throw constituent $x$ away if $p(x) < 100 \times p(y)$ for some $y$ that spans the same set of words.

- Throw away $NP_{i,j}$ because $p(NP_{i,j}) = 0$.
More strategies

Heuristics
Within the CYK algorithm, heuristically throw away constituents that probably won’t make it into a complete parse. In other words limit the number of nodes saved in each cell of the CYK table.

Where $x$ and $y$ are constituents:

- Throw constituent $x$ away if $p(x) < 10^{-200}$.
- Throw $x$ away if $p(x) < 100 \times p(y)$ for some $y$ that spans the same set of words.
- Throw away $NP_{i,j}$ b/c $p(NP_{i,j}) = 0.00002571$, and $p(VP_{i,j}) = 0.0003211$
Dealing w. sentence length

Since the EVALB package doesn’t evaluate sentences greater than length 40, there’s no need to attempt to parse them.

Add a step to your algorithm to calculate the length of the input sentence, and then to return a blank line for sentences with length greater than 40.

Thus, we reduce $n$, but of course this doesn’t really improve the parser, just the eval numbers.
## Effects of sentence length

Collins Parser results (Collins 1997) for words $\leq 40$

<table>
<thead>
<tr>
<th>labeled recall</th>
<th>label precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>88.1</td>
<td>88.6</td>
</tr>
</tbody>
</table>

Collins Parser results (Collins 1997) for words $\leq 100$

<table>
<thead>
<tr>
<th>labeled recall</th>
<th>label precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>87.5</td>
<td>88.1</td>
</tr>
</tbody>
</table>