Practical Linguistically Motivated Parsing
with Combinatory Categorial Grammar

Stephen Clark

University of Cambridge Computer Laboratory

JHU Language Technology Summer School, June 2009
Natural Language Parsing

- Automatically assigning structure to a natural language input
- More specifically, taking a sentence as input and, using a pre-defined grammar, assigning some structure to it
Phrase Structure

S
 / 
NP  VP
 /  /
DT  NP  RB  VP
   /   /
the  JJ  NNS  MD
   /   /
proposed  changes  would
   /    /
   VP  VP
   /    /
 VB  VB
  /    /
allow  allow
   /     /
NP  S
 /  /
NNS  VP
 /  /
executives  TO
 /  to
   VP
   /  /
VP  VP
 /  /
VB  VB
  /  /
report  report
   /   /
NNS  NNS
   /   /
exercises  early
   /     /
   ADVP
   /     /
RB  CONJP
  /  /
early  CC  RB
 /  /
and  often
 /  /
NP  NP
 /  /
of  of
 /  /
NNS  NNS
 /  /
options  options

Stephen Clark

Practical Linguistically Motivated Parsing

JHU, June 2009
Dependency Structure

John hit the ball with the bat
From 1953 to 1955, 9.8 billion Kent cigarettes with the filters were sold, the company said.

<table>
<thead>
<tr>
<th>x1</th>
<th>x2 x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>company(x1)</td>
<td>say(x2)</td>
</tr>
<tr>
<td>single(x1)</td>
<td>agent(x2,x1)</td>
</tr>
<tr>
<td></td>
<td>theme(x2,x3)</td>
</tr>
<tr>
<td></td>
<td>proposition(x3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>card(x4)=billion</td>
</tr>
<tr>
<td>9.8(x4)</td>
</tr>
<tr>
<td>kent(x4)</td>
</tr>
<tr>
<td>cigarette(x4)</td>
</tr>
<tr>
<td>plural(x4)</td>
</tr>
<tr>
<td>event(x6)</td>
</tr>
<tr>
<td>event(x2)</td>
</tr>
</tbody>
</table>
Why Build these Structures?

- We want to know the meaning of the sentence
- Structured representations allow us to access the semantics
- Who did What to Whom
Applications

- Question Answering/Semantic Search
- Machine Translation
- Information Extraction
- Dialogue Systems
- ...
Today’s Tutorial

• Part I
  • why is automatic parsing difficult?
  • Combinatory Categorial Grammar

• Part II
  • parsing with CCG
  • statistical parsing models
  • parsing the web
Why is Automatic Parsing Difficult?

- Obtaining a *wide-coverage* grammar which can handle arbitrary real text is challenging
Why is Automatic Parsing Difficult?

- Obtaining a *wide-coverage* grammar which can handle arbitrary real text is challenging
- Natural language is surprisingly *ambiguous*
Syntactic Ambiguity

John saw the man with the telescope.

S
NP
John
VP
V
saw
NP
the
PP
with
NP
the
telescope
Ambiguity: the problem is worse than you think
Ambiguity: the problem is worse than you think
Ambiguity: the problem is even worse than that

- Put the block in the box on the table 2 analyses
Ambiguity: the problem is even worse than that

- Put the block in the box on the table 2 analyses
- Put the block in the box on the table beside the chair 5 analyses
Ambiguity: the problem is even worse than that

- Put the block in the box on the table 2 analyses
- Put the block in the box on the table beside the chair 5 analyses
- Put the block in the box on the table beside the chair before the table 14 analyses
Ambiguity: the problem is even worse than that

- Put the block in the box on the table 2 analyses
- Put the block in the box on the table beside the chair 5 analyses
- Put the block in the box on the table beside the chair before the table 14 analyses
- Put the block in the box on the table beside the chair before the table in the kitchen 42 analyses
Ambiguity: the problem is even worse than that

- Put the block in the box on the table 2 analyses
- Put the block in the box on the table beside the chair 5 analyses
- Put the block in the box on the table beside the chair before the table 14 analyses
- Put the block in the box on the table beside the chair before the table in the kitchen 42 analyses
- ... 132 analyses
- ... 469 analyses
- ... 1430 analyses
- ... 4862 analyses
Ambiguity: the problem is even worse than that

- Wider grammar coverage $\Rightarrow$ more analyses
- In practice this could mean millions (or more) of parses for a single sentence
- We need a *parse model* giving the goodness of each parse
- We need an efficient representation of the large parse space, and an efficient way to search it
Grammars for Natural Language Parsing

- Standard approach is to use a Context Free Grammar

\[
S \rightarrow NP \ VP
\]
\[
VP \rightarrow V \ NP, \ V \ NP \ PP
\]
\[
PP \rightarrow P \ NP
\]
\[
NP \rightarrow DT \ N
\]
\[
DT \rightarrow the, \ a
\]
\[
N \rightarrow cat, \ dog
\]
\[
V \rightarrow chased, \ jumped
\]
\[
P \rightarrow over
\]
Combinatory Categorial Grammar (CCG)

- Categorial grammar (CG) is one of the oldest grammar formalisms (Ajdukiewicz, 1935; Bar-Hillel, 1953; Lambek 1958)
- Various flavours of CG now available: type-logical CG, algebraic pre-groups (Lambek), CCG
- CCG is now an established linguistic formalism (Steedman, 1996, 2000)
  - syntax; semantics; prosody and information structure; wide-coverage parsing; generation
  - http://groups.inf.ed.ac.uk/ccg/index.html
Combinatory Categorial Grammar (CCG)

- CCG is a *lexicalised grammar*
- An elementary syntactic structure – for CCG a *lexical category* – is assigned to each word in a sentence

\[ \text{walked: } S \backslash NP \text{ ‘give me an NP to my left and I return a sentence’} \]
Combinatory Categorial Grammar (CCG)

- CCG is a *lexicalised grammar*

- An elementary syntactic structure – for CCG a *lexical category* – is assigned to each word in a sentence

  *walked*: $S \backslash NP$ ‘give me an NP to my left and I return a sentence’

- A small number of rules define how categories can combine – rules based on the *combinators* from Combinatory Logic
CCG Lexical Categories

- Atomic categories: \( S, N, NP, PP, \ldots \) (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
CCG Lexical Categories

- Atomic categories: $S$, $N$, $NP$, $PP$, ... (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
- Complex categories encode subcategorisation information
  - intransitive verb: $S/\NP$ walked
  - transitive verb: $(S\NP)/NP$ respected
  - ditransitive verb: $((S\NP)/NP)/NP$ gave
CCG Lexical Categories

- Atomic categories: $S$, $N$, $NP$, $PP$, … (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
- Complex categories encode subcategorisation information
  - intransitive verb: $S\backslash NP$ walked
  - transitive verb: $(S\backslash NP)/NP$ respected
  - ditransitive verb: $((S\backslash NP)/NP)/NP$ gave
- Complex categories can encode modification
  - PP nominal: $(NP\backslash NP)/NP$
  - PP verbal: $((S\backslash NP)\backslash (S\backslash NP))/NP$
interleukin – 10 inhibits production

NP (S\NP)/NP NP
A Simple CCG Derivation

\[ \text{interleukin} - 10 \quad \text{inhibits} \quad \text{production} \]

\[ \frac{\text{NP}}{(S\backslash NP)/\text{NP}} \quad \frac{\text{NP}}{S\backslash \text{NP}} \]

> forward application
A Simple CCG Derivation

interleukin – 10
\[
\text{NP} \quad \text{inhibits} \quad \text{production}
\]
\[
(S\backslash NP)/NP \quad \text{NP}
\]
\[
S\backslash NP \quad >
\]
\[
S \quad <
\]

> forward application
< backward application
Function Application Rule Schemata

- Forward (>) and backward (<) application:

\[ X/Y \quad Y \quad \Rightarrow \quad X \quad (>) \]
\[ Y\quad X\backslash Y \quad \Rightarrow \quad X \quad (<) \]
Classical Categorial Grammar

- ‘Classical’ Categorial Grammar only has application rules
- Classical Categorial Grammar is context free

Diagram:

```
S
├── S\NP
│   ├── NP
│   │   └── interleukin-10
│   ├── (S\NP)/NP
│   │   └── inhibits
│   └── NP
    └── production
```
Classical Categorial Grammar

- ‘Classical’ Categorial Grammar only has application rules
- Classical Categorial Grammar is context free

interleukin-10 inhibits production

S
   VP
      NP V NP
interleukin-10 inhibits production
Extraction out of a Relative Clause

<table>
<thead>
<tr>
<th>The</th>
<th>company</th>
<th>which</th>
<th>Microsoft</th>
<th>bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP/N</td>
<td>N</td>
<td>(NP\NP)/(S/NP)</td>
<td>NP</td>
<td>(S\NP)/NP</td>
</tr>
</tbody>
</table>
Extraction out of a Relative Clause

\[
\begin{array}{cccc}
\text{The} & \text{company} & \text{which} & \text{Microsoft bought} \\
NP/N & N & (NP\backslash NP)/(S/NP) & NP \\
S/(S\backslash NP) & (S\backslash NP)/NP \\
\end{array}
\]

\[\rightarrow T\quad \text{type-raising}\]
Extraction out of a Relative Clause

The company which Microsoft bought

NP/N  N  (NP\NP)/(S/NP)  NP  (S\NP)/NP

S/(S\NP)  \rightarrow^{T}  S/NP

\rightarrow^{B}

> T  type-raising
> B  forward composition
Extraction out of a Relative Clause

\[
\begin{align*}
\text{The} & \quad \text{company} & \quad \text{which} & \quad \text{Microsoft} & \quad \text{bought} \\
NP/N & \quad N & \quad (NP\backslash NP)/(S/NP) & \quad NP & \quad (S\backslash NP)/NP \\
S/(S\backslash NP) & \quad \Rightarrow^T & \quad S/NP & \quad \Rightarrow^B & \quad NP\backslash NP
\end{align*}
\]
Extraction out of a Relative Clause

The company which Microsoft bought

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$

$\text{NP} / \text{N} \quad \text{N} \quad (\text{NP} \backslash \text{NP}) / (\text{S} / \text{NP})$
Forward Composition and Type-Raising

- Forward composition ($\succ_B$):

  $X / Y \ Y / Z \Rightarrow X / Z$ ($\succ_B$)

- Type-raising ($T$):

  $X \Rightarrow T/(T\backslash X)$ ($\succ_T$)
  $X \Rightarrow T\backslash(T/X)$ ($<_T$)

- Extra combinatory rules increase the weak generative power to mild context-sensitivity
“Non-constituents” in CCG – Right Node Raising

Google
\[
\begin{array}{c}
\text{NP} \\
S/(S\backslash NP)
\end{array}
\]
\[
\frac{S/(S\backslash NP)}{NP}
\]
\[
\frac{S/(S\backslash NP)}{T}
\]
sells
\[
\frac{(S\backslash NP)/NP}{NP}
\]
but
\[
\frac{conj}{NP}
\]
Microsoft
\[
\frac{NP}{NP}
\]
\[
\frac{(S\backslash NP)/NP}{NP}
\]
\[
\frac{S/(S\backslash NP)}{T}
\]
\[
\frac{S/(S\backslash NP)}{NP}
\]
\[
\frac{NP}{NP}
\]
\[
\frac{S/(S\backslash NP)}{T}
\]
\[
\text{type-raising}
\]
“Non-constituents” in CCG – Right Node Raising

Google sells but Microsoft buys shares

\[
\begin{align*}
\text{Google} & \quad \text{sells} & \quad \text{but} & \quad \text{Microsoft} & \quad \text{buys} & \quad \text{shares} \\
\text{NP} & \quad (S\backslash NP)/NP & \quad \text{conj} & \quad \text{NP} & \quad (S\backslash NP)/NP & \quad \text{NP} \\
S/(S\backslash NP) & \quad \xrightarrow{\mathbf{T}} & \quad S/\mathbf{NP} & \quad \xrightarrow{\mathbf{B}} & \quad S/(S\backslash NP) & \quad \xrightarrow{\mathbf{T}} & \quad S/\mathbf{NP} & \quad \xrightarrow{\mathbf{B}}
\end{align*}
\]

> \textbf{T} \quad \text{type-raising} \\
> \textbf{B} \quad \text{forward composition}
“Non-constituents” in CCG – Right Node Raising

Google \(\text{sells}\) but Microsoft \(\text{buys}\) shares

\[
\begin{align*}
\text{NP} & \quad (S\backslash NP)/NP \\
S/(S\backslash NP) & \quad \text{NP} \\
S/\text{NP} & \quad (S\backslash NP)/NP \\
S/(S\backslash NP) & \quad \text{NP} \\
S/\text{NP} & \quad <\Phi>
\end{align*}
\]
“Non-constituents” in CCG – Right Node Raising

Google sells but Microsoft buys shares

NP \( (S\backslash NP)/NP \) \( \text{conj} \) NP \( (S\backslash NP)/NP \) NP

\[ S/(S\backslash NP) \] \( \Rightarrow^T \)
\[ S/\text{NP} \] \( \Rightarrow^B \)
\[ S/\text{NP} \] <\( \Phi \)>

\[ \Rightarrow \]

Stephen Clark
Practical Linguistically Motivated Parsing
JHU, June 2009
Combinatory Categorial Grammar

- **CCG** is *mildly* context sensitive
- Natural language is provably non-context free
- Constructions in Dutch and Swiss German (Shieber, 1985) require more than context free power for their analysis
  - these have *crossing* dependencies (which **CCG** can handle)

```
Type 0 languages
|   |
Context sensitive languages
|   | Mildly context sensitive languages = natural languages (?)
Context free languages
|   |
Regular languages
```
Grammar Engineering vs. Grammar Extraction

• How can we obtain the wide-coverage grammar?
  • a syntactician writes the rules (whilst consulting corpus data)
  • a syntactician annotates sentences with grammatical structures, and the grammar is read automatically off that
  • the grammar is induced automatically from raw text
Grammar Engineering vs. Grammar Extraction

- How can we obtain the wide-coverage grammar?
  - a syntactician writes the rules (whilst consulting corpus data)
  - a syntactician annotates sentences with grammatical structures, and the grammar is read automatically off that
  - the grammar is induced automatically from raw text
- Introduces a level of modularity into the process:

  linguist | computer scientist
The Penn Treebank (1993)

- 40,000 sentences (1M words) of English newspaper text annotated with phrase-structure trees
- Took annotators at the University of Pennsylvania 3 years to build
- Has been very influential (dominant) in parsing and NLP research
A PTB Phrase-Structure Tree

S
├── NP
│   └── It
│       ├── V
│       │   └── ADJP
│       │       └── ADJ
│       │           └── S
│       │               └── NP
│       │                   └── VP
│       │                         └── V
│       │                             └── TO
│       │                                 └── VP
│       │                                     └── SBAR
│       │                                           └── WHNP
│       │                                               └── S
│       │                                                   └── NP
│       │                                                       └── VP
│       │                                                           └── V
│       │                                                               └── TO
│       │                                                                   └── VP
│       │                                                                           └── NP
│       │                                                                               └── VP
│       │                                                                                      └── NP
│       │                                                                                                          └── V
│       │                                                                                                                └── TO
│       │                                                                                                                        └── VP
│       │                                                                                                                                └── NP
│       │                                                                                                                                        └── V
│       │                                                                                                                                                    └── VP
│       │                                                                                                                                                        └── NP
A **CCG** Treebank: CCGbank

- CCGbank developed by Hockenmaier and Steedman (Hockenmaier, 2003)
- Phrase-structure trees in Penn Treebank (semi-)automatically converted into **CCG** derivations
- But note phrase-structure trees not isomorphic to **CCG** analyses (e.g. coordination)
A **CCG** Derivation Tree

```
S[dcl]
  /
(S[dcl]\NP)/(S[to]\NP)
     /
NP    (S[to]\NP)/(S[to]\NP)/NP
     |    |
Marks persuades

S[to]\NP
  /
NP   S[b]\NP
     |
  |   to
  |   merge

Brooks to merge
```

Stephen Clark

Practical Linguistically Motivated Parsing

JHU, June 2009
Inducing a Grammar

- Grammar (lexicon) can be read off the leaves of the trees
- In addition to the grammar, CCGbank provides training data for the statistical models
Inducing a Grammar

• \( \approx 1200 \) lexical category types in CCGbank
  (compared with 45 \textit{POS} tags in Penn Treebank)
• Frequency cut-off of 10 gives \( \approx 400 \) types (when applied to sections 2-21 of CCGbank)
  • this set has very high coverage on unseen data (section 00)
• In addition to the grammar, CCGbank provides training data for the statistical models
Parsing with CCG

• **Stage 1**
  • Assign POS tags and lexical categories to words in the sentence
  • Use taggers to assign the POS tags and categories
    – based on standard Maximum Entropy tagging techniques
Parsing with CCG

- **Stage 1**
  - Assign POS tags and lexical categories to words in the sentence
  - Use taggers to assign the POS tags and categories
    - based on standard Maximum Entropy tagging techniques
- **Stage 2**
  - Combine the categories using the combinatory rules
  - Can use standard bottom-up CKY chart-parsing algorithm
Parsinig with CCG

• **Stage 1**
  • Assign POS tags and lexical categories to words in the sentence
  • Use taggers to assign the POS tags and categories
    – based on standard Maximum Entropy tagging techniques

• **Stage 2**
  • Combine the categories using the combinatory rules
  • Can use standard bottom-up CKY chart-parsing algorithm

• **Stage 3**
  • Find the highest scoring derivation according to some model
    – e.g. generative model, CRF, perceptron
  • Viterbi algorithm finds this efficiently
Maximum Entropy Tagging

• Consider POS tagging as an example
• 45 POS tags from the Penn Treebank
Maximum Entropy Tagging (Ratnaparkhi, 1998)

- Use local log-linear models to estimate $P(tag|context)$:

$$P(t|x) = \frac{1}{Z_x} e^{\sum_j \lambda_j f_j(t,x)}$$

$Z_x$ is a normalisation constant ensuring a proper distribution

- Conditional probability of tag sequence:

$$P(t_1, t_2, \ldots, t_n|w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(t_i|x_i)$$
Feature-Based Tagging

- Context is a 5-word window surrounding target word
- Features are the words in the window, plus the two previously assigned tags
- Additional features for rare and unknown words
  - suffix information
  - is the word capitalised?
  - does the word contain a hyphen?
Features in Log-Linear Tagging Models

- Features are binary-valued indicator functions
- Contextual predicates identify elements of the context which may be useful for predicting the tag

\[ f_i(t, x) = \begin{cases} 
1 & \text{if } \text{word}(x) = \text{the} \quad \& \quad t = \text{det} \\
0 & \text{otherwise}
\end{cases} \]

- \( \text{word}(x) = \text{the} \) is an example of a contextual predicate
- Features can be arbitrary properties of the context
- No requirement for the features to be independent
- Variety of training algorithms available to automatically set the weights
Supertagging

He goes on the road with his piano

A bitter conflict with global implications

NP (S[dcl]\NP)/PP PP/NP NP/N N ((S\NP)\(S\NP))/NP NP/N N

NP/N N/N N (NP\NP)/NP N/N N

Stephen Clark

Practical Linguistically Motivated Parsing

JHU, June 2009
CCG Supertagging

<table>
<thead>
<tr>
<th>He</th>
<th>goes</th>
<th>on</th>
<th>the</th>
<th>road</th>
<th>with</th>
<th>his</th>
<th>piano</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>(S[dcl]\NP)/PP</td>
<td>PP/NP</td>
<td>NP/N</td>
<td>N</td>
<td>((S\NP)(S\NP))/NP</td>
<td>NP/N</td>
<td>N</td>
</tr>
</tbody>
</table>

- ≈ 400 lexical category types
- Baseline tagging accuracy is ≈ 72%
  - baseline is to assign tag most frequently seen with word in training data, and assign N to unseen words
- Baseline for Penn Treebank POS tagging is ≈ 90%
Lexical Category Sequence for Newspaper Sentence

\[ \text{In|IN} \quad \text{an|DT} \quad \text{Oct.|NNP} \quad 19|CD \quad \text{review|NN} \quad \text{of|IN} \quad \text{The|DT} \]

\[ \text{(S/S)/NP} \quad \text{NP[nb]/N} \quad \text{N/N} \quad \text{N/N} \quad \text{N} \quad \text{(NP\NP)/NP} \quad \text{NP[nb]/N} \]

\[ \text{Misanthrope|NNP} \quad \text{at|IN} \quad \text{Chicago|NNP} \quad \text{’s|POS} \quad \text{Goodman|NNP} \quad \text{Theatre|NNP} \]

\[ \text{N} \quad \text{(NP\NP)/NP} \quad \text{N} \quad \text{(NP[nb]/N)\NP} \quad \text{N/N} \quad \text{N} \]

\[ \text{−LRB−|LRB} \quad \text{Revitalized|JJ} \quad \text{Classics|NNS} \quad \text{Take|VBZ} \quad \text{the|DT} \quad \text{Stage|NN} \quad \text{...} \]

\[ \text{(NP\NP)/S[dcl]} \quad \text{N/N} \quad \text{N} \quad \text{(S[dcl]\NP)/NP} \quad \text{NP[nb]/N} \quad \text{N} \]
A Maximum Entropy Supertagger

- Maximum Entropy tagging method can be applied to CCG supertagging
- Features are the words and POS tags in the 5-word window, plus the two previously assigned categories
- Per-word tagging accuracy is $\approx 92\%$
- This accuracy is not high enough for the tagger to serve as an effective front-end to a CCG parser
  - roughly two errors per WSJ sentence on average
Multitagging

- Potentially assign more than one category to a word
  - assign all categories whose probability is within some factor $\beta$ of the highest probability category
- Accuracy is over 97% at only 1.4 categories per word
- Accuracy is now high enough to serve as a front-end to the parser
Chart Parsing

- A chart is just a tabular data structure which stores the constituents spanning each subsequence of words
- The chart can be filled in “bottom-up”
  - start by combining lexical categories and continue to apply the combinatory rules until the whole sentence is covered
- Fill in the cells corresponding to the shortest subsequences first:
  - the *CKY algorithm*
Chart Parsing

The chart-parsing algorithm operates bottom-up. Packing the chart efficiently represents a large derivation space.
• **CKY** chart-parsing algorithm operates bottom-up
• **CKY** chart-parsing algorithm operates bottom-up

• *Packing* the chart efficiently represents a large derivation space
CKY Algorithm

chart[i][j] is a cell containing categories spanning words from i to i + j

initialise chart with categories of span 1 (lexical categories)

LOOP over span of result category (j = 2 to SENT_LENGTH)
  LOOP over start position of left combining category (i = 0 to SENT_LENGTH - j)
    LOOP over span of left combining category (k = 1 to j - 1)
      chart[i][j] ++ Combine(chart[i][k], chart[i + k][j - k])
• DP algorithms can be run over the packed representation
• The Viterbi algorithm finds the highest scoring derivation
Linear Parsing Model

$$\text{Score}(d, S) = \sum_i \lambda_i f_i(d) = \bar{\lambda} \cdot \phi(d)$$

- Features are counts over $d$
  - root category of $d$ (plus lexical head)
  - $\langle$lexical category, lexical item$\rangle$ pairs
  - rule feature: $S \rightarrow NP \ S \backslash NP$ (plus lexical head)
  - predicate argument dependency: subj(bought, IBM) (plus distance)
  - “Backing-off” features with words replaced by POS tags
- Use Perceptron training to set the weights
Training Data from CCGbank

subj(persuades, Marks)
obj(persuades, Brooks)
subj(merge, Brooks)
to-inf(persuades, merge)
Feature Representation

\[
f_i : D \rightarrow \mathcal{N} \quad (3\,000\,000 \leq i \leq 1)
\]
Linear Parsing Model

\[ \text{Score}(d, s) = \sum_{i} \lambda_i \cdot f_i(d) = \overline{\lambda} \cdot \overline{f}(d) \]

- \( f_i \) are the features (defined by hand)
- \( \lambda_i \) are the corresponding weights (which need to be learned)
Perceptron Training

\[
\text{Score}(d, S) = \sum_i \lambda_i f_i(d) = \bar{\lambda} \cdot \phi(d)
\]

**Inputs:** training examples \((x_i, y_i)\)

**Initialisation:** set \(\bar{\lambda} = 0\)

**Algorithm:**

\[
\text{for } t = 1..T, \ i = 1..N \\
\text{calculate } z_i = \arg \max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \bar{\lambda} \\
\text{if } z_i \neq y_i \\
\quad \bar{\lambda} = \bar{\lambda} + \Phi(x_i, y_i) - \Phi(x_i, z_i)
\]

**Outputs:** \(\bar{\lambda}\)
**Perceptron Training**

\[ W_0 = <0, 0, 0, ..., 0, 0, 0, ..., 0> \]
Perceptron Training

\[ W_0 = \langle 0, 0, 0, \ldots, 0, 0, \ldots, 0, 0, 0, 0, \ldots, 0 \rangle \]

- \( f_{15}, f_{21}, f_{56}, f_{120}, f_{212}, f_{348}, f_{419} \)
- \( f_{19}, f_{25}, f_{75}, f_{150}, f_{211}, f_{346}, f_{450}, f_{500}, f_{525} \)
- \( f_{1}, f_{20}, f_{55}, f_{100}, f_{210}, f_{345} \)

**SENT1:**

- \( w_1 \)
- \( w_2 \)
- \( w_3 \)
- \( w_4 \)
- \( w_5 \)
Perceptron Training (Online)

UPDATE WEIGHTS:

W1 = <0,1,0,...,-1,0,...,-1,0,1,0,-1,...,0>

f1, f20, f55, f100, f210, f345
f19, f25, f75, f150, f211, f346, f450, f500, f525
f15, f21, f56, f120, f212, f348, f419
Perceptron Training

\[ W_1 = <0,1,0,...,-1,0,...,-1,...0,1,0,-1,...,0> \]
Perceptron Training

W1 = <0,1,0,...,-1,0,...,-1,...0,1,0,-1,...,0>

f11, f21, f57, f90, f145, f250
f21, f25, f76, f151, f222, f348, f444, f507, f575
f17, f45, f155, f167, f678

 SENT2:

DECODE:

1
2
3
4

S
S/NP
S/S
S
S
S/S
(S/NP)/PP
PP/NP
PP/NP
(NP\NP)/NP
(S\NP)/PP
NP/PP
(S\NP)/NP
PP/PP
NP
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
N
### Perceptron Training

#### UPDATE WEIGHTS:

<table>
<thead>
<tr>
<th>4</th>
<th>S/S</th>
<th>S/NP</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>S/NP</td>
<td>(S/NP)/PP</td>
<td>(S/NP)/NP</td>
</tr>
<tr>
<td>2</td>
<td>S/S</td>
<td>S/NP</td>
<td>PP/NP</td>
</tr>
<tr>
<td>1</td>
<td>NP</td>
<td>(S/NP)/NP</td>
<td>NP</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>(S/NP)/PP</td>
<td>PP/NP</td>
</tr>
<tr>
<td></td>
<td>S/(S/NP)</td>
<td>(NP\NP)/NP</td>
<td>(NP\NP)/NP</td>
</tr>
</tbody>
</table>

**SENT2:**

<table>
<thead>
<tr>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>w4</th>
</tr>
</thead>
</table>

\[ W2 = <0, 2, -1, ..., -1, 1, ..., -1, 0, 1, 0, -2, ..., -1> \]

- f11, f21, f57, f90, f145, f250
- f21, f25, f76, f151, f222, f348, f444, f507, f575
- f17, f45, f155, f167, f678
Perceptron Training is Expensive

\[ \text{Score}(d|S) = \sum_i \lambda_i f_i(d) = \bar{\lambda} \cdot \phi(d) \]

**Inputs:** training examples \((x_i, y_i)\)

**Initialisation:** set \(\bar{\lambda} = 0\)

**Algorithm:**

for \(t = 1..T, i = 1..N\)

- calculate \(z_i = \arg \max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \bar{\lambda}\)
- if \(z_i \neq y_i\)

\[ \bar{\lambda} = \bar{\lambda} + \Phi(x_i, y_i) - \Phi(x_i, z_i) \]

**Outputs:** \(\bar{\lambda}\)

- Requires an efficient decoder
Efficient Decoding with CCG

• Supertagging leaves decoder with (relatively) little left to do
• Each packed chart needs at most 20 MB RAM
• Most probable derivation can be found very quickly with Viterbi
• Training takes 5 hours for 10 iterations
Parser Evaluation

- Compare output of the parser with a *gold standard*
- Exact match metric sometimes used but a little crude
- Partial match against a set of *grammatical relations* currently the method of choice
  - measures recovery of semantically important relations
  - relatively theory-neutral representation
Head-Based Grammatical Relations

- *She gave the present to Kim*
  (ncsubj gave She _)
  (dobj gave present)
  (iobj gave to)
  (dobj to Kim)
  (det present the)
Head-Based Grammatical Relations

- *She gave the present to Kim*
  (ncsubj gave She _)
  (dobj gave present)
  (iobj gave to)
  (dobj to Kim)
  (det present the)

- *The company wants to wean itself away from expensive gimmicks*
  (xcomp to wants wean)
  (iobj wean from)
  (ncmodprt wean away)
  (dobj wean itself)
  (dobj from gimmicks)
  (ncmod_ gimmicks expensive)
  ...
### Mapping CCG Dependencies to GRs

- Argument slots in CCG dependencies are mapped to GRs

<table>
<thead>
<tr>
<th>CCG lexical category</th>
<th>arg slot</th>
<th>GR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S[dcl]\NP₁)/NP₂</td>
<td>1</td>
<td>(nsubj %l %f)</td>
</tr>
<tr>
<td>(S[dcl]\NP₁)/NP₂</td>
<td>2</td>
<td>(dobj %l %f)</td>
</tr>
<tr>
<td>(NP\NP₁)/NP₂</td>
<td>1</td>
<td>(prep %f %l)</td>
</tr>
<tr>
<td>(NP\NP₁)/NP₂</td>
<td>2</td>
<td>(pobj %l %f)</td>
</tr>
<tr>
<td>NP[nb]/N₁</td>
<td>1</td>
<td>(det %f %l)</td>
</tr>
</tbody>
</table>

- Mapping is many-to-many
Test Suite: DepBank

- 700 sentences of newspaper text manually annotated with GRs
- Calculate precision and recall over GRs

\[
\text{Prec} = \frac{\# \text{ correct}}{\# \text{ proposed by parser}} \quad \text{Rec} = \frac{\# \text{ correct}}{\# \text{ in gold standard}}
\]

\[
F\text{-score} = \frac{2 \cdot P \cdot R}{P + R}
\]

Stephen Clark

Practical Linguistically Motivated Parsing

JHU, June 2009
Final Parsing Results

<table>
<thead>
<tr>
<th>Prec</th>
<th>Rec</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>84.1</td>
<td>82.8</td>
<td>83.4</td>
</tr>
</tbody>
</table>

- These scores compare favourably with the best results in the literature on this test set
## Results by Dependency Type

<table>
<thead>
<tr>
<th>GR</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncssubj</td>
<td>79.6</td>
</tr>
<tr>
<td>dobj</td>
<td>87.7</td>
</tr>
<tr>
<td>obj2</td>
<td>66.7</td>
</tr>
<tr>
<td>iobj</td>
<td>73.4</td>
</tr>
<tr>
<td>clausal</td>
<td>75.0</td>
</tr>
<tr>
<td>ncmmod</td>
<td>76.1</td>
</tr>
<tr>
<td>aux</td>
<td>92.8</td>
</tr>
<tr>
<td>det</td>
<td>95.1</td>
</tr>
<tr>
<td>conj</td>
<td>77.5</td>
</tr>
</tbody>
</table>
Parsing the Web

• Why parse the Web?
  • semantic search
  • provide massive amounts of data for knowledge acquisition
  • ...

• Need a fast parser (to process billions of web pages)
• Need a parser that isn’t overly tuned to newspaper text
Speed Demo

- Use of the CCG supertagger (and some highly engineered C++) leads to a highly efficient linguistically motivated parser
- Can process **1 billion words** in less than 5 days with 18 machines
- Can we make the parser go faster still?
Conclusion

- Robust linguistically-motivated parsing of real text is now possible
  - but can it really help in NLP applications?
- What’s left?
  - plenty of room for accuracy improvements
  - cheap ways to get more training data