Algorithms for NLP

Parsing V

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley
Binarization / Markovization

NP

DT JJ NN NN

v=1, h=∞

NP

DT @NP[DT]

JJ @NP[DT,JJ]

NN @NP[DT,JJ,NN]

NN

NP

DT @NP[DT]

JJ @NP[...JJ]

NN @NP[...NN]

NN

NP

DT @NP

JJ @NP

NN @NP

NN
Binarization / Markovization

\[
\begin{align*}
\text{NP} & \rightarrow \text{DT} \quad \text{JJ} \quad \text{NN} \quad \text{NN} \\
\text{v}=2, h=\infty & \\
\text{NP} & \rightarrow \text{DP} \quad \text{NN} \\
\text{v}=2, h=1 & \\
\text{NP} & \rightarrow \text{DP} \quad \text{NN} \\
\text{v}=2, h=0 & \\
\text{NP} & \rightarrow \text{DP} \\
\end{align*}
\]
Grammar Projections

Coarse Grammar

\[
\begin{align*}
\text{NP} & \rightarrow \text{DT} \ @\text{NP} \\
\text{NP} & \rightarrow \text{JJ} \ @\text{NP} \\
\text{NP} & \rightarrow \text{NN} \ @\text{NP} \\
\text{NP} & \rightarrow \text{NN} \ @\text{NP} \\
\end{align*}
\]

Fine Grammar

\[
\begin{align*}
\text{NP}^\text{VP} & \rightarrow \text{DT}^\text{NP} \ @\text{NP}^\text{VP}[\text{DT}] \\
\text{NP}^\text{VP} & \rightarrow \text{JJ}^\text{NP} \ @\text{NP}^\text{VP}[...,\text{JJ}] \\
\text{NP}^\text{VP} & \rightarrow \text{NN}^\text{NP} \ @\text{NP}^\text{VP}[...,\text{NN}] \\
\end{align*}
\]

\[
\begin{align*}
\text{NP} & \rightarrow \text{DT} \ @\text{NP} \\
\text{NP}^\text{VP} & \rightarrow \text{DT}^\text{NP} \ @\text{NP}^\text{VP}[\text{DT}]
\end{align*}
\]

Note: X-Bar Grammars are projections with rules like \( XP \rightarrow Y @X \) or \( XP \rightarrow @X Y \) or \( @X \rightarrow X \)
<table>
<thead>
<tr>
<th>Coarse Symbols</th>
<th>Fine Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>NP^VP</td>
</tr>
<tr>
<td></td>
<td>NP^S</td>
</tr>
<tr>
<td>@NP</td>
<td>@NP^VP[DT]</td>
</tr>
<tr>
<td></td>
<td>@NP^S[DT]</td>
</tr>
<tr>
<td>DT</td>
<td>@NP^VP[...JJ]</td>
</tr>
<tr>
<td></td>
<td>@NP^S[...JJ]</td>
</tr>
<tr>
<td></td>
<td>DT^NP</td>
</tr>
</tbody>
</table>
Efficient Parsing for Structural Annotation
Coarse-to-Fine Pruning

\[ P(X| i, j, S) < \text{threshold} \]

E.g. consider the span 5 to 12:

coarse:

fine:
Coarse-to-Fine Pruning

For each coarse chart item $X[i,j]$, compute posterior probability:

$$\frac{\alpha(X, i, j) \cdot \beta(X, i, j)}{\alpha(\text{root}, 0, n)} < \text{threshold}$$

E.g. consider the span 5 to 12:

coarse: ... CP NP VP ... 

fine: ...
Computing Marginals

\[ \alpha(X, i, j) = \sum_{X \rightarrow YZ} \sum_{k \in (i, j)} P(X \rightarrow YZ) \alpha(Y, i, k) \alpha(Z, k, j) \]
\[ \beta(X, i, j) = \sum_{Y \rightarrow ZX} \sum_{k \in [0, i]} P(Y \rightarrow ZX) \beta(Y, k, j) \alpha(\xi, k, i) \]
\[ + \sum_{Y \rightarrow XZ} \sum_{k \in (j, n]} P(Y \rightarrow XZ) \beta(Y, i, k) \alpha(Z, j, k) \]
Pruning with A*

- You can also speed up the search without sacrificing optimality
- For agenda-based parsers:
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimality [Klein and Manning 03]
Efficient Parsing for Lexical Grammars
Lexicalized Trees

- Add “head words” to each phrasal node
  - Syntactic vs. semantic heads
  - Headship not in (most) treebanks
  - Usually use head rules, e.g.:
    - **NP:**
      - Take leftmost NP
      - Take rightmost N*
      - Take rightmost JJ
      - Take right child
    - **VP:**
      - Take leftmost VB*
      - Take leftmost VP
      - Take left child
Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like
  \[ \text{VP(saw)} \rightarrow \text{VBD(saw)} \text{ NP-C(her)} \text{ NP(saw)} \]

- Never going to get these atomically off of a treebank

- Solution: break up derivation into smaller steps
Lexical Derivation Steps

- A derivation of a local tree [Collins 99]
  
  Choose a head tag and word

  Choose a complement bag

  Generate children (incl. adjuncts)

  Recursively derive children
Lexicalized CKY

bestScore(X, i, j, h)
  if (j = i+1)
    return tagScore(X, s[i])
  else
    return
      max max score(X[h]→Y[h] Z[h‘]) * bestScore(Y, i, k, h) * bestScore(Z, k, j, h‘)
      max score(X[h]→Y[h‘] Z[h]) * bestScore(Y, i, k, h‘) * bestScore(Z, k, j, h)

Quartic Parsing

- Turns out, you can do (a little) better [Eisner 99]

- Gives an $O(n^4)$ algorithm
- Still prohibitive in practice if not pruned
Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the $O(n^5)$ CKY
  - Remember only a few hypotheses for each span $<i,j>$.
  - If we keep $K$ hypotheses at each span, then we do at most $O(nK^2)$ work per span (why?)
  - Keeps things more or less cubic (and in practice is more like linear!)

- Also: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)
The Charniak parser prunes using a two-pass, coarse-to-fine approach [Charniak 97+]

- First, parse with the base grammar
- For each $X:[i,j]$ calculate $P(X|i,j,s)$
  - This isn’t trivial, and there are clever speed ups
- Second, do the full $O(n^5)$ CKY
  - Skip any $X:[i,j]$ which had low (say, $< 0.0001$) posterior
  - Avoids almost all work in the second phase!

- Charniak et al 06: can use more passes
- Petrov et al 07: can use many more passes
Results

- Some results
  - Collins 99 – 88.6 F1 (generative lexical)
  - Charniak and Johnson 05 – 89.7 / 91.3 F1 (generative lexical / reranked)
  - Petrov et al 06 – 90.7 F1 (generative unlexical)
  - McClosky et al 06 – 92.1 F1 (gen + rerank + self-train)
Latent Variable PCFGs
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson ’98]
  - Head lexicalization [Collins ’99, Charniak ’00]
  - Automatic clustering?
Latent Variable Grammars

Parse Tree

Grammar G

Lexicon

Parameters
EM algorithm:
- Brackets are known
- Base categories are known
- Only induce subcategories

Just like Forward-Backward for HMMs.
Refinement of the DT tag

- DT
  - the (0.50)
  - a (0.24)
  - The (0.08)

- DT-1
  - a (0.61)
  - the (0.19)
  - an (0.11)

- DT-2
  - the (0.80)
  - The (0.15)
  - a (0.01)

- DT-3
  - this (0.39)
  - that (0.28)
  - That (0.11)

- DT-4
  - some (0.20)
  - all (0.19)
  - those (0.12)
Hierarchical refinement

- **the (0.50)**
  - **a (0.24)**
  - **The (0.08)**
  - **that (0.15)**
  - **this (0.14)**
  - **some (0.11)**

- **the (0.54)**
  - **a (0.25)**
  - **The (0.09)**
  - **this (0.14)**
  - **that (0.28)**
  - **That (0.11)**

- **a (0.61)**
  - **the (0.19)**
  - **an (0.11)**
  - **some (0.20)**
  - **all (0.19)**
  - **those (0.12)**
Hierarchical Estimation Results

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat Training</td>
<td>87.3</td>
</tr>
<tr>
<td>Hierarchical Training</td>
<td>88.4</td>
</tr>
</tbody>
</table>
Refinement of the , tag

- Splitting all categories equally is wasteful:
- Want to split complex categories more
- Idea: split everything, roll back splits which were least useful
Adaptive Splitting Results

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>88.4</td>
</tr>
<tr>
<td>With 50% Merging</td>
<td>89.5</td>
</tr>
</tbody>
</table>

![Graph showing parsing accuracy (F1) vs. total number of grammar symbols. Three lines represent 50% Merging, Hierarchical Training, and Flat Training.]
Number of Lexical Subcategories
## Learned Splits

- **Proper Nouns (NNP):**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP-12</td>
<td>John</td>
<td>Robert</td>
<td>James</td>
</tr>
<tr>
<td>NNP-2</td>
<td>J.</td>
<td>E.</td>
<td>L.</td>
</tr>
<tr>
<td>NNP-1</td>
<td>Bush</td>
<td>Noriega</td>
<td>Peters</td>
</tr>
<tr>
<td>NNP-15</td>
<td>New</td>
<td>San</td>
<td>Wall</td>
</tr>
<tr>
<td>NNP-3</td>
<td>York</td>
<td>Francisco</td>
<td>Street</td>
</tr>
</tbody>
</table>

- **Personal pronouns (PRP):**

<table>
<thead>
<tr>
<th>PRP-0</th>
<th>it</th>
<th>He</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRP-1</td>
<td>it</td>
<td>he</td>
<td>they</td>
</tr>
<tr>
<td>PRP-2</td>
<td>it</td>
<td>them</td>
<td>him</td>
</tr>
</tbody>
</table>
**Learned Splits**

- **Relative adverbs (RBR):**

<table>
<thead>
<tr>
<th>RBR</th>
<th>further</th>
<th>lower</th>
<th>higher</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Cardinal Numbers (CD):**

<table>
<thead>
<tr>
<th>CD</th>
<th>one</th>
<th>two</th>
<th>Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1989</td>
<td>1990</td>
<td>1988</td>
</tr>
<tr>
<td>11</td>
<td>million</td>
<td>billion</td>
<td>trillion</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>78</td>
<td>58</td>
<td>34</td>
</tr>
</tbody>
</table>
# Final Results (Accuracy)

<table>
<thead>
<tr>
<th>Language</th>
<th>Method</th>
<th>≤ 40 words F1</th>
<th>all F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENG</td>
<td>Charniak&amp;Johnson ‘05 (generative)</td>
<td>90.1</td>
<td>89.6</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>90.6</td>
<td>90.1</td>
</tr>
<tr>
<td>GER</td>
<td>Dubey ‘05</td>
<td>76.3</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>80.8</td>
<td>80.1</td>
</tr>
<tr>
<td>CHN</td>
<td>Chiang et al. ‘02</td>
<td>80.0</td>
<td>76.6</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>86.3</td>
<td>83.4</td>
</tr>
</tbody>
</table>

Still higher numbers from reranking / self-training methods
Efficient Parsing for Hierarchical Grammars
Coarse-to-Fine Inference

- Example: PP attachment
Hierarchical Pruning

coarse:

split in two:

split in four:

split in eight:
Bracket Posteriors
1621 min
111 min
35 min
15 min
(no search error)
Other Syntactic Models
Dependency Parsing

- Lexicalized parsers can be seen as producing *dependency trees*

- Each local binary tree corresponds to an attachment in the dependency graph
Dependency Parsing

Pure dependency parsing is only cubic [Eisner 99]

Some work on non-projective dependencies

- Common in, e.g. Czech parsing
- Can do with MST algorithms [McDonald and Pereira 05]
Another way to derive a tree:

Parsing
- No useful dynamic programming search
- Can still use beam search [Ratnaparkhi 97]
Tree Insertion Grammars

- Rewrite large (possibly lexicalized) subtrees in a single step

- Formally, a *tree-insertion grammar*
- Derivational ambiguity whether subtrees were generated atomically or compositionally
- Most probable *parse* is NP-complete
TIG: Insertion

\[ \phi \]
\[
A \downarrow
\]

\[ \psi \]
\[
A
\]

\[ \phi' \]
\[
A
\]

\[ \psi \]

\[ S \]
\[ NP \downarrow \]
\[ VP \]
\[ V \]
\[ NP \downarrow \]
\[ saw \]

\[ NP \]
\[ D \downarrow \]
\[ N \]
\[ man \]

\[ S \]
\[ NP \]
\[ D \downarrow \]
\[ N \]
\[ man \]
\[ V \]
\[ NP \downarrow \]
\[ saw \]
Tree-adjoining grammars

- Start with *local trees*
- Can insert structure with *adjunction* operators
- Mildly context-sensitive
- Models long-distance dependencies naturally
- ... as well as other weird stuff that CFGs don’t capture well (e.g. cross-serial dependencies)
TAG: Long Distance
CCG Parsing

- **Combinatory Categorial Grammar**
  - Fully (mono-) lexicalized grammar
  - Categories encode argument sequences
  - Very closely related to the lambda calculus (more later)
  - Can have spurious ambiguities (why?)

\[
\begin{align*}
  John & \vdash NP \\
  shares & \vdash NP \\
  buys & \vdash (S\backslash NP)/NP \\
  sleeps & \vdash S\backslash NP \\
  well & \vdash (S\backslash NP)\backslash(S\backslash NP)
\end{align*}
\]

```
\begin{center}
\begin{tikzpicture}
  \node {S} [grow'=right, sibling angle=120, edge from parent fork down]
    child {node {NP} edge from parent node [left] {John}}
    child {node {S\backslash NP} [grow'=right, sibling angle=120, edge from parent fork down]
      child {node {buys}}
      child {node {shares}}
    }
\end{tikzpicture}
\end{center}
```
Empty Elements
Empty Elements
Empty Elements

- In the PTB, three kinds of empty elements:
  - Null items (usually complementizers)
  - Dislocation (WH-traces, topicalization, relative clause and heavy NP extraposition)
  - Control (raising, passives, control, shared argumentation)

- Need to reconstruct these (and resolve any indexation)
Example: English

Farmers was quick *ICH*-2 yesterday *-3 TO point out the problems it sees *T*-1.
Example: German
## Types of Empties

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>POS</th>
<th>Label</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>NP</td>
<td>*</td>
<td>18,334</td>
<td>NP trace (e.g., <em>Sam was seen</em>)</td>
</tr>
<tr>
<td>WHNP</td>
<td>NP</td>
<td><em>T</em></td>
<td>8,620</td>
<td>WH trace (e.g., <em>the woman</em> you saw <em>T</em>)</td>
</tr>
<tr>
<td></td>
<td>NP</td>
<td><em>U</em></td>
<td>7,478</td>
<td>Empty units (e.g., $25 <em>U</em>)</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td><em>T</em></td>
<td>5,635</td>
<td>Empty complementizers (e.g., <em>Sam said 0 Sasha snores</em>)</td>
</tr>
<tr>
<td>WHADVP</td>
<td>ADVP</td>
<td><em>T</em></td>
<td>4,063</td>
<td>Moved clauses (e.g., <em>Sam had to go, Sasha explained</em> <em>T</em>)</td>
</tr>
<tr>
<td>WHNP</td>
<td>SBAR</td>
<td></td>
<td>2,033</td>
<td>Empty clauses (e.g., <em>Sam had to go, Sasha explained (SBAR)</em></td>
</tr>
<tr>
<td>WHADVP</td>
<td>0</td>
<td></td>
<td>1,759</td>
<td>Empty relative pronouns (e.g., <em>the woman 0 we saw</em>)</td>
</tr>
<tr>
<td>WHADVP</td>
<td>0</td>
<td></td>
<td>575</td>
<td>Empty relative pronouns (e.g., <em>no reason 0 to leave</em>)</td>
</tr>
</tbody>
</table>

![Diagram of linguistic structures](image.png)
A Pattern-Matching Approach

- [Johnson 02]
Pattern-Matching Details

- Something like transformation-based learning
- Extract patterns
  - Details: transitive verb marking, auxiliaries
  - Details: legal subtrees
- Rank patterns
  - Pruning ranking: by correct / match rate
  - Application priority: by depth
- Pre-order traversal
- Greedy match
# Top Patterns Extracted

<table>
<thead>
<tr>
<th>Count</th>
<th>Match</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>5816</td>
<td>6223</td>
<td>(S (NP (-NONE- *)) VP)</td>
</tr>
<tr>
<td>5605</td>
<td>7895</td>
<td>(SBAR (-NONE- 0) S)</td>
</tr>
<tr>
<td>5312</td>
<td>5338</td>
<td>(SBAR WHNP-1 (S (NP (-NONE- <em>T</em>-1)) VP))</td>
</tr>
<tr>
<td>4434</td>
<td>5217</td>
<td>(NP QP (-NONE- <em>U</em>))</td>
</tr>
<tr>
<td>1682</td>
<td>1682</td>
<td>(NP $ CD (-NONE- <em>U</em>))</td>
</tr>
<tr>
<td>1327</td>
<td>1593</td>
<td>(VP VBN,t (NP (-NONE- *)) PP)</td>
</tr>
<tr>
<td>700</td>
<td>700</td>
<td>(ADJP QP (-NONE- <em>U</em>))</td>
</tr>
<tr>
<td>662</td>
<td>1219</td>
<td>(SBAR (WHNP-1 (-NONE- 0)) (S (NP (-NONE- <em>T</em>-1)) VP))</td>
</tr>
<tr>
<td>618</td>
<td>635</td>
<td>(S S-1 , NP (VP VBD (SBAR (-NONE- 0) (S (-NONE- <em>T</em>-1)))))) .)</td>
</tr>
<tr>
<td>499</td>
<td>512</td>
<td>(SINV `` S-1 , '' (VP VBZ (S (-NONE- <em>T</em>-1))) NP .)</td>
</tr>
<tr>
<td>361</td>
<td>369</td>
<td>(SINV `` S-1 , '' (VP VBD (S (-NONE- <em>T</em>-1))) NP .)</td>
</tr>
<tr>
<td>352</td>
<td>320</td>
<td>(S NP-1 (VP VBZ (S (NP (-NONE- *-1)) VP))))</td>
</tr>
<tr>
<td>346</td>
<td>273</td>
<td>(S NP-1 (VP AUX (VP VBN,t (NP (-NONE- *-1)) PP))))</td>
</tr>
<tr>
<td>322</td>
<td>467</td>
<td>(VP VBD,t (NP (-NONE- *)) PP)</td>
</tr>
<tr>
<td>269</td>
<td>275</td>
<td>(S `` S-1 , '' NP (VP VBD (S (-NONE- <em>T</em>-1)))) .)</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Empty node POS</th>
<th>Label</th>
<th>Section 23</th>
<th>Parser output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Overall)</td>
<td></td>
<td>0.93 0.83 0.88</td>
<td>0.85 0.74 0.79</td>
</tr>
<tr>
<td>NP</td>
<td>*</td>
<td>0.95 0.87 0.91</td>
<td>0.86 0.79 0.82</td>
</tr>
<tr>
<td>NP</td>
<td><em>T</em></td>
<td>0.93 0.88 0.91</td>
<td>0.85 0.77 0.81</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.94 0.99 0.96</td>
<td>0.86 0.89 0.88</td>
</tr>
<tr>
<td></td>
<td><em>U</em></td>
<td>0.92 0.98 0.95</td>
<td>0.87 0.96 0.92</td>
</tr>
<tr>
<td>S</td>
<td><em>T</em></td>
<td>0.98 0.83 0.90</td>
<td>0.97 0.81 0.88</td>
</tr>
<tr>
<td>ADVP</td>
<td><em>T</em></td>
<td>0.91 0.52 0.66</td>
<td>0.84 0.42 0.56</td>
</tr>
<tr>
<td>SBAR</td>
<td></td>
<td>0.90 0.63 0.74</td>
<td>0.88 0.58 0.70</td>
</tr>
<tr>
<td>WHNP</td>
<td>0</td>
<td>0.75 0.79 0.77</td>
<td>0.48 0.46 0.47</td>
</tr>
</tbody>
</table>
Semantic Roles
Semantic Role Labeling (SRL)

- Characterize clauses as *relations with roles*:

  \[ \text{Judge She } \text{blames } [\text{Evaluator the Government }] [\text{Reason for failing to do enough to help}] . \]

  Holman would characterise this as *blaming* [\text{Evaluator the poor}] .

  The letter quotes Black as saying that [\text{Judge white and Navajo ranchers}] misrepresent their livestock losses and *blame* [\text{Reason everything}] [\text{Evaluator on coyotes}] .

- Says more than which NP is the subject (but not much more):
- Relations like *subject* are syntactic, relations like *agent* or *message* are semantic
- Typical pipeline:
  - Parse, then label roles
  - Almost all errors locked in by parser
  - Really, SRL is quite a lot easier than parsing
He heard the sound of liquid slurping in a metal container as Farrell approached him from behind.
- FrameNet: roles shared between verbs
- PropBank: each verb has its own roles
- PropBank more used, because it’s layered over the treebank (and so has greater coverage, plus parses)
- Note: some linguistic theories postulate fewer roles than FrameNet (e.g. 5-20 total: agent, patient, instrument, etc.)
PropBank Example

fall.01

sense: move downward
roles: Arg1: thing falling
Arg2: extent, distance fallen
Arg3: start point
Arg4: end point

Sales fell to $251.2 million from $278.7 million.

arg1: Sales
rel: fell
arg4: to $251.2 million
arg3: from $278.7 million
rotate.02  sense: shift from one thing to another
roles:  Arg0:  causer of shift
       Arg1:  thing being changed
       Arg2:  old thing
       Arg3:  new thing

Many of Wednesday’s winners were losers yesterday as investors quickly took profits and rotated their buying to other issues, traders said. (wsj_1723)
arg0:  investors
rel:  rotated
arg1:  their buying
arg3:  to other issues
**PropBank Example**

**aim.01**

- **sense:** intend, plan
- **roles:**
  - Arg0: aimer, planner
  - Arg1: plan, intent

The Central Council of Church Bell Ringers aims *trace* to improve relations with vicars. 
(Anonymous, 2008)

- arg0: The Central Council of Church Bell Ringers
- rel: aims
- arg1: *trace* to improve relations with vicars

**aim.02**

- **sense:** point (weapon) at
- **roles:**
  - Arg0: aimer
  - Arg1: weapon, etc.
  - Arg2: target

Banks have been aiming packages at the elderly.

- arg0: Banks
- rel: aiming
- arg1: packages
- arg2: at the elderly
(NP-SBJ (JJ massive) (JJ internal) (NN debt) )
  (VP (VBZ has))
  (VP (VBN forced))
  (S)
  (NP-SBJ-1 (DT the) (NN government) )
  (VP)
  (VP (TO to))
  (VP (VB borrow))
  (ADVP-MNR (RB massively))...
Path Features

![Diagram of syntactic tree with labeled nodes: S, NP, PRP, VP, VB, DT, NN, He, ate, some, pancakes.]

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB↑VP↓PP</td>
<td>PP argument/adjunct</td>
</tr>
<tr>
<td>VB↑VP↑S↓NP</td>
<td>subject</td>
</tr>
<tr>
<td>VB↑VP↑NP</td>
<td>object</td>
</tr>
<tr>
<td>VB↑VP↑VP↑S↓NP</td>
<td>subject (embedded VP)</td>
</tr>
<tr>
<td>VB↑VP↓ADVP</td>
<td>adverbiaial adjunct</td>
</tr>
<tr>
<td>NN↑NP↑NP↑PP</td>
<td>prepositional complement of noun</td>
</tr>
</tbody>
</table>
Results

- **Features:**
  - Path from target to filler
  - Filler’s syntactic type, headword, case
  - Target’s identity
  - Sentence voice, etc.
  - Lots of other second-order features

- **Gold vs parsed source trees**
  - SRL is fairly easy on gold trees
  - Harder on automatic parses

<table>
<thead>
<tr>
<th></th>
<th>CORE</th>
<th>ARGM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F1</td>
<td>Acc.</td>
</tr>
<tr>
<td></td>
<td>92.2</td>
<td>80.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CORE</th>
<th>ARGM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F1</td>
<td>Acc.</td>
</tr>
<tr>
<td></td>
<td>84.1</td>
<td>66.5</td>
</tr>
</tbody>
</table>