Transforming
Projective Bilexical Dependency Grammars
into efficiently-parsable CFGs
with Unfold-Fold

Mark Johnson
Microsoft Research    Brown University

ACL 2007
Motivation and summary

- What’s the relationship between CKY parsing and the Eisner/Satta $O(n^3)$ PBDG parsing algorithm? (c.f., McAllester 1999)
  - *split-head encoding*, collecting left and right dependents separately
  - *unfold-fold transform* reorganizes grammar for efficient CKY parsing
- Approach generalizes to 2nd-order dependencies
  - predict argument given governor and sibling (McDonald 2006)
  - predict argument given governor and governor’s governor
- *In principle* can use any CFG parsing or estimation algorithm for PBDGs
  - transformed grammars typically too large to enumerate
  - my CKY implementations transform grammar on the fly
Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

Conclusion
Projective Bilexical Dependency Grammars

- Projective Bilexical Dependency Grammar (PBDG)

  \[
  \begin{array}{c}
  0 \leadsto \text{gave} \\
  \text{gave} \leadsto \text{dog} \\
  \text{gave} \leadsto \text{bone} \\
  \text{Sandy} \leadsto \text{gave} \\
  \text{the} \leadsto \text{dog} \\
  \text{a} \leadsto \text{bone}
  \end{array}
  \]

- A dependency parse generated by the PBDG

  0 Sandy gave the dog a bone

- Weights can be attached to dependencies (and preserved in CFG transforms)
A naive encoding of PBDGs as CFGs

\[
\begin{align*}
S & \rightarrow X_u \\
X_u & \rightarrow u \\
X_u & \rightarrow X_v X_u \\
X_u & \rightarrow X_u X_v
\end{align*}
\]

where $0 \rightarrow u$

$\vdash$

$\vdash$

$\vdash$

S

X

gave

\[
\begin{align*}
X_{\text{Sandy}} & \rightarrow X \\
X_{\text{Sandy}} & \rightarrow \text{gave} \\
\text{gave} & \rightarrow X_{\text{the}} \\
\text{the} & \rightarrow \text{dog} \\
\text{dog} & \rightarrow X_{\text{dog}} \\
\text{dog} & \rightarrow a \\
a & \rightarrow X_{\text{bone}} \\
\text{bone} & \rightarrow X_{\text{bone}}
\end{align*}
\]
Spurious ambiguity in naive encoding

- Naive encoding allows dependencies on different sides of head to be freely reordered

⇒ Spurious ambiguity in CFG parses (not present in PBDG parses)
Parsing naive CFG encoding takes $O(n^5)$ time

- A production schema such as

$$X_u \rightarrow X_u X_v$$

has 5 variables, and so can match input in $O(n^5)$ different ways.
Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

Conclusion
Simple split-head encoding

- Replace input word $u$ with a left variant $u_\ell$ and a right variant $u_r$ (can be avoided in practice with fancy book-keeping)

$$\text{Sandy gave the dog a bone} \Downarrow \text{Sandy}_\ell \text{ Sandy}_r \text{ gave}_\ell \text{ gave}_r \text{ the}_\ell \text{ the}_r \text{ dog}_\ell \text{ dog}_r \text{ a}_\ell \text{ a}_r \text{ bone}_\ell \text{ bone}_r$$

- PCFG separately collects left dependencies and right dependencies

$$S \rightarrow X_u \quad \text{where } 0 \xrightarrow{\sim} u$$

$$X_u \rightarrow L_u \ L \ R \quad \text{where } u \in \Sigma$$

$$L_u \rightarrow u_\ell$$

$$L_u \rightarrow X_v \ L_u \quad \text{where } v \xleftarrow{\sim} u$$

$$R \rightarrow u_r$$

$$R \rightarrow u \ X_v \quad \text{where } u \xleftarrow{\sim} v$$
Simple split-head CFG parse
$L_u$ and $R_u$ heads are phrase-peripheral $\Rightarrow O(n^4)$

- Heads of $L_u$ and $R_u$ are always at right (left) edge

```
S → X_u where 0 ↘ u
X_u → L_u R where $u \in \Sigma$
L_u → $u_l$
L_u → X_v L_u where $v ↘ u$
R → $u_r$
R → R X_v where $u ↘ v$
```

- $X_u \rightarrow L_u R$ take $O(n^3)$
- $R_u \rightarrow R X_v$ take $O(n^4)$
Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

Conclusion
The Unfold-Fold transform

- Unfold-fold originally proposed for transforming recursive programs; used here to transform CFGs into new CFGs
- *Unfolding* a nonterminal replaces it with its expansion

\[
\begin{align*}
A & \rightarrow \alpha B \gamma \\
B & \rightarrow \beta_1 \\
B & \rightarrow \beta_2 \\
\ldots
\end{align*}
\Rightarrow
\begin{align*}
A & \rightarrow \alpha \beta_1 \gamma \\
A & \rightarrow \alpha \beta_2 \gamma \\
B & \rightarrow \beta_1 \\
B & \rightarrow \beta_2 \\
\ldots
\end{align*}
\]

- *Folding* is the inverse of unfolding (replace RHS with nonterminal)

\[
\begin{align*}
A & \rightarrow \alpha B \gamma \\
B & \rightarrow \beta \\
\ldots
\end{align*}
\Rightarrow
\begin{align*}
A & \rightarrow \alpha B \gamma \\
B & \rightarrow \beta \\
\ldots
\end{align*}
\]

- Transformed grammar generates same language (Sato 1992)
Unfold-fold converts $O(n^4)$ to $O(n^3)$ grammar

- Unfold $X_v$ responsible for $O(n^4)$ parse time

  $$
  \begin{align*}
  L_u & \rightarrow \ u_l \\
  L_u & \rightarrow \ X_v \ L_u \\
  X_v & \rightarrow \ L_v \ R \\
  \end{align*}
  \Rightarrow
  \begin{align*}
  L_u & \rightarrow \ u_l \\
  L_u & \rightarrow \ L_v \ R \ L_u \\
  \end{align*}

- Introduce new non-terminals $x M_y$ (doesn’t change language)

  $$
  x M_y \rightarrow x R \ L_y
  $$

- Fold two children of $L_u$ into $x M_y$

  $$
  \begin{align*}
  L_u & \rightarrow \ u_l \\
  L_u & \rightarrow \ L_v \ R \ L_u \\
  x M_y & \rightarrow x R \ L_y \\
  \end{align*}
  \Rightarrow
  \begin{align*}
  L_u & \rightarrow \ u_l \\
  L_u & \rightarrow \ L_v \ R \ L_u \\
  x M_y & \rightarrow x R \ L_y \\
  \end{align*}
  \Rightarrow
  \begin{align*}
  L_u & \rightarrow \ u_l \\
  L_u & \rightarrow \ L_v \ R \ L_u \\
  x M_y & \rightarrow x R \ L_y \\
  \end{align*}$$
Transformed grammar collects left and right dependencies separately

- $X_v$ constituents (which cause $O(n^4)$ parse time) no longer used
- Head annotations now all phrase peripheral $\Rightarrow O(n^3)$ parse time
- Dependencies can be recovered from parse tree
- Basically same as Eisner and Satta $O(n^3)$ algorithm
  - explains why Inside-Outside sanity check fails for Eisner/Satta
  - two copies of each terminal $\Rightarrow$ each terminals’ Outside probability is double the Inside sentence probability
Parse using $O(n^3)$ transformed split-head grammar

0 Sandy gave the dog a bone
Parsing time of CFG encodings of same PBDG

<table>
<thead>
<tr>
<th>CFG schemata</th>
<th>sentences parsed / second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive (O(n^5)) CFG</td>
<td>45.4</td>
</tr>
<tr>
<td>(O(n^4)) simple split-head CFG</td>
<td>406.2</td>
</tr>
<tr>
<td>(O(n^3)) transformed split-head CFG</td>
<td>3580.0</td>
</tr>
</tbody>
</table>

- Weighted PBDG; all pairs of heads have some dependency weight
- Dependency weights precomputed before parsing begins
- Timing results on a 3.6GHz Pentium 4 machine parsing section 24 of the PTB
- CKY parsers with grammars hard-coded in C (no rule lookup)
- Dependency accuracy of Viterbi parses = 0.8918 for all grammars

*Feature extraction is much slower than even naive CFG*
Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

Conclusion
Predict argument based on governor and sibling

- Very similar to second-order algorithm given by McDonald (2006)
Because left and right dependencies are assembled separately, only captures 2nd-order dependencies where one dependency is leftward and other is rightward.
Outline

Projective Bilexical Dependency Grammars

Simple split-head encoding

$O(n^3)$ split-head CFGs via Unfold-Fold

Transformations capturing 2nd-order dependencies

Conclusion
Conclusion and future work

- Presented a reduction from PBDGs to $O(n^3)$ parsable CFGs
  - split-head CFG representation of PBDGs
  - Unfold-fold transform
- CKY algorithm on resulting CFG simulates Eisner/Satta algorithm on original PBDG
- Makes CFG techniques applicable to PBDGs
  - max marginal parsing (Goodman 1996) and other CFG parsing and estimation algorithms
- Can capture different dependencies, yielding different PDG models
  - 2nd-order “horizontal” dependencies (McDonald 2006)
  - what other combinations of dependencies can we capture? (if we permit $O(n^4)$ parse time?)
  - do any of these improve parsing accuracy?