Motivation

Parsing is crucial in NLP

• Syntactic parsing can allow for superior performance
  • Machine Translation
  • Information Retrieval
  • Sentiment Analysis

• Parsing is still far from perfect
  • Too slow for web-scale text and not accurate enough

• Incremental nature of shift-reduce parsing allows for new features that could help improve speed and accuracy
Combinatory Categorial Grammar (CCG)

- CCG is a lexicalised grammar formalism (Steedman, 2000)
  - Each word has a category dictating its behaviour
  - Categories are combined using a small set of combinatory rules
- Complex categories are functions that takes a category as an argument and returns another category

\[
\begin{align*}
\text{Jack} & \quad \text{saw} & \quad \text{money} \\
\text{NP} & \quad (S \backslash \text{NP})/\text{NP} & \quad \text{NP} \\
\hline
S & \quad \text{NP} & \quad \text{NP} \\
\hline
S & \quad \text{NP} & \quad <
\end{align*}
\]
The C&C parser (Clark and Curran, 2007) is a state-of-the-art CCG parser

- Primary focus on speed, accuracy and coverage
- Achieves over 100 sentences/second using the CKY algorithm

Training and testing occur on CCGbank, a corpus of 40,000 annotated sentences (Hockenmaier and Steedman, 2007)

Parsing pipeline is currently linear – no interaction
Supertagging for Efficient CCG Parsing

- Naïvely could apply every possible CCG category to each word
  - There are 1,286 different CCG categories in CCGbank 02-21
- Supertagging $\rightarrow$ eliminate unlikely CCG categories
The horse jumps on the hill

DT NN VBZ IN DT NN

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

N/N (S/NP)/NP PP/NP

N (NP\NP)/NP
CCG Categories used for the Final Parse

The horse jumps on the hill

\[ NP/N \rightarrow DT \quad NN \quad VBZ \quad IN \quad DT \quad NN \]

\[ N \rightarrow (S\backslash NP)/PP \qquad PP/NP \rightarrow NP/N \rightarrow NP \rightarrow PP \rightarrow S\backslash NP \rightarrow S \]
Improving Supertagging

- The supertagger currently receives no data from the parser
  - Kummerfeld et al. (2010) adapted the supertagger to the parser, improving parsing speed significantly
- Optimal: Parser assists supertagger by providing a partial understanding of the sentence
Incremental Parsing for Higher Accuracy

- Humans analyse sentences incrementally to assist with understanding upcoming words (Pickering, 1999; Tanenhaus and Brown-Schmidt, 2008)
- Incremental parsing allows for a partial derivation to develop without all words being supplied
- Can perform POS/super tagging decisions when parser already understands earlier part of sentence → higher accuracy
Constituent Parsing using the CKY Algorithm

- Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1965; Younger, 1967) is a chart parsing algorithm
- Dynamic programming (DP) over the chart allows efficient computation but does not allow incremental parsing
Shift-Reduce Algorithm

- Shift-reduce algorithm allows for \textit{incremental} parsing
- Popular for programming language parsing (unambiguous)
- With ambiguous grammars, \textit{worst-case is exponential}

- Shift-reduce parsing implemented in two CCG parsers:
  - Deterministic CCG (Hassan et al., 2008) \textit{Restricted expressive power and low accuracy}
  - Shift-reduce CCG parser (Zhang and Clark, 2011) \textit{Competitive but aggressive beam pruning for practical speeds}
- What if we want to explore the full search space with SR?
Shift-Reduce Example for CCG

Parsing “Jack saw money”

∅
Shift-Reduce Example for CCG

Parsing “Jack saw money”

∅ ← NP (Jack)
Shift-Reduce Example for CCG

Parsing “Jack saw money”

∅ ← NP (Jack) ← (S\NP)/NP (saw)
Shift-Reduce Example for CCG

Parsing “Jack saw money”

∅ ← NP (Jack) ← (S\NP)/NP (saw) ← NP (money)
Shift-Reduce Example for CCG

Parsing “Jack saw money”

∅ ← NP (Jack) ← S\NP
Shift-Reduce Example for CCG

Parsing “Jack saw money”

∅ ← S
Shift-Reduce $\rightarrow$ Exponential

$\emptyset \leftarrow A \leftarrow B \leftarrow C \leftarrow D \leftarrow E$

Reduction Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$D \ E$</td>
</tr>
<tr>
<td>$G$</td>
<td>$D \ E$</td>
</tr>
<tr>
<td>$H$</td>
<td>$C \ D \ E$</td>
</tr>
</tbody>
</table>
Shift-Reduce $\rightarrow$ Exponential

$\emptyset \leftarrow A \leftarrow B \leftarrow C \leftarrow D \leftarrow E$

$\emptyset \leftarrow A \leftarrow B \leftarrow C \leftarrow F$

$\emptyset \leftarrow A \leftarrow B \leftarrow C \leftarrow G$

$\emptyset \leftarrow A \leftarrow B \leftarrow H$

Reduction Rules

- $F \leftarrow D \ E$
- $G \leftarrow D \ E$
- $H \leftarrow C \ D \ E$

Merity et al. Frontier Pruning for Shift-Reduce CCG Parsing December, 2011
Graph-Structured Stack (GSS)

- GSS (Tomita, 1988) allows for polynomial shift-reduce parsing by performing dynamic programming.
- Not explored extensively, implemented in only two parsers.
- GSS has never been implemented for CCG.
- Based around three concepts to improve efficiency:
  - Splitting
  - Combining
  - Local Ambiguity Packing
Graph-Structured Stack (GSS)

\[
\emptyset \leftarrow A \leftarrow B \leftarrow C \leftarrow D \leftarrow E \leftarrow H
\]

**Reduction Rules**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>←</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>G</td>
<td>←</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>H</td>
<td>←</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Merity et al.  
Frontier Pruning for Shift-Reduce CCG Parsing  
December, 2011
Graph-Structured Stack (GSS)

Reduction Rules

\[
F \leftarrow D \ E \\
G \leftarrow D \ E \\
H \leftarrow C \ D \ E
\]
Graph-Structured Stack (GSS)

Reduction Rules

\[
\begin{align*}
J & \leftarrow F \quad I \\
J & \leftarrow G \quad I
\end{align*}
\]
Results for the Graph-Structured Stack in CCG Parsing

- First time a GSS for CCG parsing has been implemented
- Polynomial instead of exponential in the worst-case
- GSS-based SR and CKY algorithms can be compared

<table>
<thead>
<tr>
<th>Parser</th>
<th>Coverage (%)</th>
<th>Labeled F-score (%)</th>
<th>Speed (sents/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKY C&amp;C Gold POS</td>
<td>99.34</td>
<td>86.79</td>
<td>96.3</td>
</tr>
<tr>
<td>SR C&amp;C Gold POS</td>
<td>99.58</td>
<td>86.78</td>
<td>71.3</td>
</tr>
<tr>
<td>CKY C&amp;C Auto POS</td>
<td>99.25</td>
<td>84.59</td>
<td>82.0</td>
</tr>
<tr>
<td>SR C&amp;C Auto POS</td>
<td>99.50</td>
<td>84.53</td>
<td>61.2</td>
</tr>
</tbody>
</table>

Table: Final evaluation of the CKY and SR CCG parsers on Section 23 of CCGbank (Auto indicates automatically assigned POS tags were used)
Frontier Features

- As the parser is incremental, we can represent the current parser state using frontier features
- A *frontier* is all possible CCG derivations at a given point

```
I saw John with binoculars
```

Merity et al. Frontier Pruning for Shift-Reduce CCG Parsing December, 2011
The search space for parsers is massive
  - Pruning removes unlikely states from the search space

Frontier features allow the pruning classifier to better understand where the partial sentence could lead

For training, we use unpruned parser output
  - Identify only the nodes used in the final parse

During parsing, we discard any unlikely derivations resulting in improved parsing speed

The classifier used is an online binary perceptron classifier
  - Potential for future work in self training
Frontier Features for Pruning

\[ \emptyset \leftarrow NP \leftarrow S \backslash NP \leftarrow (NP \backslash NP) / NP \]

\[ \emptyset \leftarrow NP \leftarrow S \backslash NP \leftarrow ((S \backslash NP) \backslash (S \backslash NP)) / NP \]

I saw John with binoculars

Merity et al. Frontier Pruning for Shift-Reduce CCG Parsing December, 2011
Features for Frontier Pruning

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>$S \backslash NP$</td>
</tr>
<tr>
<td>Binary Composition</td>
<td>$(S \backslash NP)/NP$ and $NP$</td>
</tr>
<tr>
<td>Forward Application</td>
<td>True</td>
</tr>
<tr>
<td>Head Word</td>
<td>saw</td>
</tr>
<tr>
<td>Head POS</td>
<td>VBD</td>
</tr>
</tbody>
</table>

| Previous Frontier      | $NP$               |
| Next Frontier          | $((S \backslash NP)\backslash(S \backslash NP))/NP$ |
| Next Frontier          | $(NP \backslash NP)/NP$ |

```
I saw John with binoculars
```

Merity et al. Frontier Pruning for Shift-Reduce CCG Parsing December, 2011
Improving Recall of the Marked Set

- Averaged pruned tree size is 6.7% of original
- Recall of marked set is only 72.9%
- If the marked set is pruned, accuracy may be impacted
- Binary perceptron classifier returns true if \( w \cdot x > 0 \)
- Improve recall by modifying the threshold level (\( \lambda \))
- \( w \cdot x > \lambda \)
- This trades accuracy for recall by increasing false positives

\[
\begin{array}{c|c|c}
\lambda_2 & \lambda_1 & 0 \\
\end{array}
\]
Improving Recall of the Marked Set

<table>
<thead>
<tr>
<th>Model</th>
<th>Coverage (%)</th>
<th>lf (%)</th>
<th>uf (%)</th>
<th>Speed (sents/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKY C&amp;C</td>
<td>99.01</td>
<td>86.37</td>
<td>92.56</td>
<td>55.6</td>
</tr>
<tr>
<td>SR C&amp;C</td>
<td>98.90</td>
<td>86.35</td>
<td>92.44</td>
<td>48.6</td>
</tr>
<tr>
<td>FP $\lambda = 0$</td>
<td>99.01</td>
<td>86.11</td>
<td>92.25</td>
<td><strong>61.1</strong></td>
</tr>
<tr>
<td>FP $\lambda = -1$</td>
<td><strong>99.06</strong></td>
<td>86.16</td>
<td>92.23</td>
<td>56.4</td>
</tr>
<tr>
<td>FP $\lambda = -2$</td>
<td>99.01</td>
<td>86.13</td>
<td>92.19</td>
<td>53.9</td>
</tr>
<tr>
<td>FP $\lambda = -3$</td>
<td><strong>99.06</strong></td>
<td>86.15</td>
<td>92.21</td>
<td>49.0</td>
</tr>
</tbody>
</table>

Table: Development tests on Section 00 of CCGbank
### Results for Frontier Pruning

<table>
<thead>
<tr>
<th>Parser</th>
<th>Coverage (%)</th>
<th>Labeled F-score (%)</th>
<th>Speed (sents/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKY</td>
<td>99.34</td>
<td>86.79</td>
<td>96.3</td>
</tr>
<tr>
<td>SR</td>
<td>99.58</td>
<td>86.78</td>
<td>71.3</td>
</tr>
<tr>
<td>FP SR</td>
<td>99.38</td>
<td>86.51</td>
<td>95.4</td>
</tr>
<tr>
<td>CKY</td>
<td>99.25</td>
<td>84.59</td>
<td>82.0</td>
</tr>
<tr>
<td>SR</td>
<td>99.50</td>
<td>84.53</td>
<td>61.2</td>
</tr>
<tr>
<td>FP SR</td>
<td>99.29</td>
<td>84.29</td>
<td>84.9</td>
</tr>
</tbody>
</table>

**Table:** Final evaluation on Section 23 of CCGbank
Conclusion

- Developed an incremental shift-reduce CCG parser
  - Extended GSS to allow for CCG parsing – first time in literature
    Worst-case polynomial instead of exponential time
  - Allows for comparison between CKY and SR algorithms
    Shift-reduce parser 34% slower than CKY parser

- Incremental parsing allows for novel features
  - Frontier pruning improves parsing speed by 39%
    Frontier pruned SR parser is slightly faster than CKY parser
Future Work

- Starting point for exploration of frontier features
- Preliminary results show substantial improvements in supertagging accuracy by providing frontier features
- Integration of pipeline components → increased accuracy
- What other tasks could benefit from direct parser interaction?
Acknowledgments

- The authors would like to thank the anonymous reviewers for their insightful comments
- This work was supported by Australian Research Council Discovery grants DP1097291 and the Capital Markets Cooperative Research Centre
- The first author was supported by a University of Sydney Merit Scholarship


References III

