Frontier Pruning for Shift-Reduce CCG Parsing

Stephen Merity and James R. Curran
ə-lab, School of Information Technologies
University of Sydney

December, 2011

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

$$
\begin{array}{ccc}
\text{Jack} & \text{saw} & \text{money} \\
\hline
NP & (S\backslash NP)/NP & NP \\
 & \multicolumn{2}{c}{\underline{\qquad\qquad\qquad\qquad}>} \\
 & \multicolumn{2}{c}{S\backslash NP} \\
\multicolumn{3}{c}{\underline{\qquad\qquad\qquad\qquad\qquad\qquad}<} \\
\multicolumn{3}{c}{S}
\end{array}
$$

The C&C parser

- 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

PoS Tagging → Supertagging → Parsing → Scoring → Decoding

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

CCG Supertagging

The	horse	jumps	on	the	hill
DT	NN	VBZ	IN	DT	NN
NP/N	N	$(S\backslash NP)/PP$	$((S\backslash NP)\backslash(S\backslash NP))/NP$	NP/N	N
	N/N	$(S\backslash NP)/NP$	PP/NP		
		N	$(NP\backslash NP)/NP$		

CCG Categories used for the Final Parse

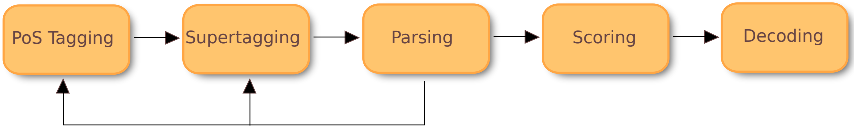
The	horse	jumps	on	the	hill
DT	NN	VBZ	IN	DT	NN
NP/N	N	$(S\backslash NP)/PP$	PP/NP	NP/N	N
NP (>)				NP (>)	
			PP (>)		
		$S\backslash NP$ (>)			
S (<)					

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

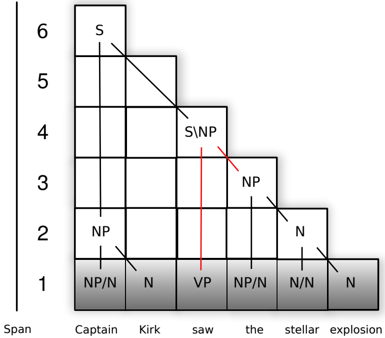
Incremental Parsing for Higher Accuracy

- 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 *incremental* parsing
- Popular for programming language parsing (unambiguous)
- With ambiguous grammars, worst-case is exponential
- Shift-reduce parsing implemented in two CCG parsers:
  - Deterministic CCG (Hassan et al., 2008)
    Restricted expressive power and low accuracy
  - Shift-reduce CCG parser (Zhang and Clark, 2011)
    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"

$$\emptyset$$

Shift-Reduce Example for CCG

Parsing "Jack saw money"

$$\emptyset \longleftarrow NP\ (Jack)$$

Shift-Reduce Example for CCG

Parsing "Jack saw money"

$$\emptyset \longleftarrow \mathit{NP}\ (\mathit{Jack}) \longleftarrow (S\backslash \mathit{NP})/\mathit{NP}\ (\mathit{saw})$$

Shift-Reduce Example for CCG

Parsing “Jack saw money”

$$\emptyset \longleftarrow NP\ (Jack) \longleftarrow \mathbf{(S\backslash NP)/NP}\ (saw) \longleftarrow \mathbf{NP}\ (money)$$

Shift-Reduce Example for CCG

Parsing "Jack saw money"

$\emptyset \longleftarrow$ **NP** *(Jack)* $\longleftarrow$ **S\NP**

Shift-Reduce Example for CCG

Parsing "Jack saw money"

$$\emptyset \longleftarrow S$$

Shift-Reduce $\rightarrow$ Exponential

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

Reduction Rules				
F	$\leftarrow$	D	E	
G	$\leftarrow$	D	E	
H	$\leftarrow$	C	D	E

Shift-Reduce $\rightarrow$ Exponential

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

$$\emptyset \longleftarrow A \longleftarrow B \longleftarrow C \longleftarrow \mathbf{F}$$

$$\emptyset \longleftarrow A \longleftarrow B \longleftarrow C \longleftarrow \mathbf{G}$$

$$\emptyset \longleftarrow A \longleftarrow B \longleftarrow \mathbf{H}$$

Reduction Rules				
F	$\leftarrow$	D	E	
G	$\leftarrow$	D	E	
H	$\leftarrow$	C	D	E

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

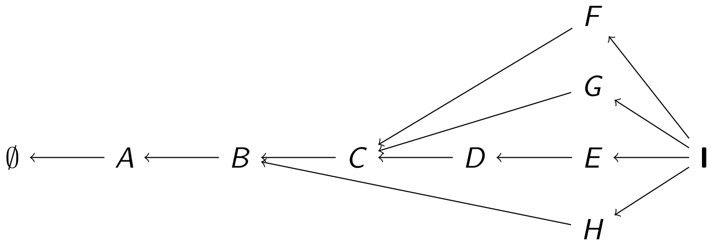
Graph-Structured Stack (GSS)

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

F → C, **G** → C, **H** → B

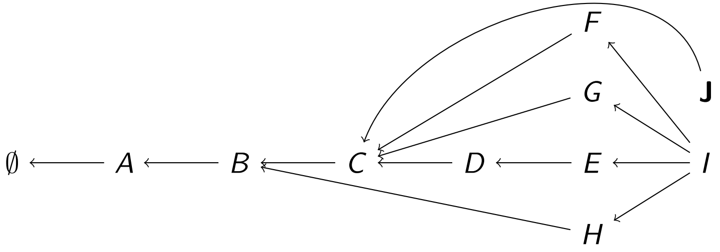
Reduction Rules				
F	$\leftarrow$	D	E	
G	$\leftarrow$	D	E	
H	$\leftarrow$	C	D	E

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			
J	$\leftarrow$	F	I
J	$\leftarrow$	G	I

Results for the Graph-Structured Stack in CCG Parsing

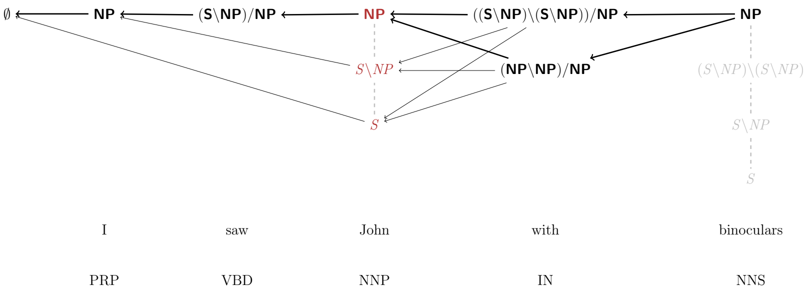
- First time a GSS for CCG parsing has been implemented
  - *Polynomial* instead of *exponential* in the worst-case
  - GSS-based SR and CKY algorithms can be compared

Parser	Coverage (%)	Labeled F-score (%)	Speed (sents/sec)
CKY C&C *Gold* POS	99.34	**86.79**	**96.3**
SR C&C *Gold* POS	**99.58**	86.78	71.3
CKY C&C *Auto* POS	99.25	**84.59**	**82.0**
SR C&C *Auto* POS	**99.50**	84.53	61.2

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



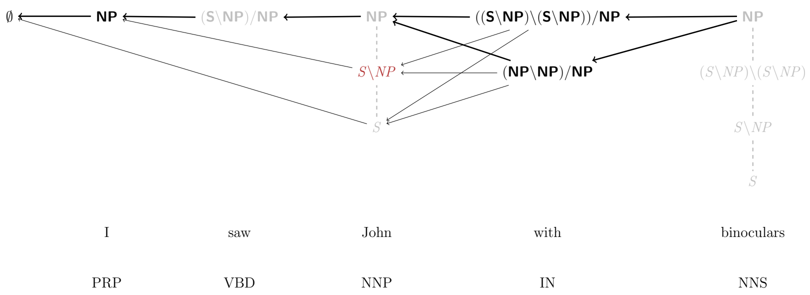
Frontier Pruning

- 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

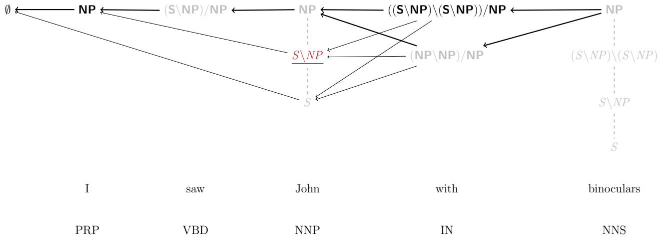
$$\emptyset \longleftarrow NP \longleftarrow S\backslash NP \longleftarrow (NP\backslash NP)/NP$$

$$\emptyset \longleftarrow NP \longleftarrow S\backslash NP \longleftarrow ((S\backslash NP)\backslash(S\backslash NP))/NP$$



Features for Frontier Pruning

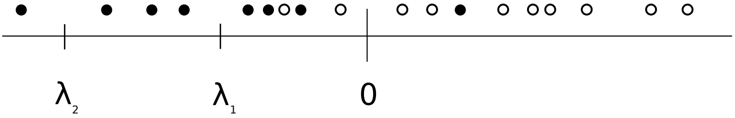
Feature Type	Example
Category	$S\backslash NP$
Binary Composition	$(S\backslash NP)/NP$ and NP
Forward Application	True
Head Word	*saw*
Head POS	*VBD*
Previous Frontier	NP
Next Frontier	$((S\backslash NP)\backslash(S\backslash NP))/NP$
Next Frontier	$(NP\backslash NP)/NP$



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 (λ)
  $w \cdot x > \lambda$
- This trades accuracy for recall by increasing false positives



Improving Recall of the Marked Set

Model	Coverage (%)	lf. (%)	uf. (%)	Speed (sents/sec)
CKY C&C	99.01	**86.37**	**92.56**	55.6
SR C&C	98.90	86.35	92.44	48.6
FP $\lambda = 0$	99.01	86.11	92.25	**61.1**
FP $\lambda = -1$	**99.06**	86.16	92.23	56.4
FP $\lambda = -2$	99.01	86.13	92.19	53.9
FP $\lambda = -3$	**99.06**	86.15	92.21	49.0

Table: Development tests on Section 00 of CCGbank

Results for Frontier Pruning

Parser	Coverage (%)	Labeled F-score (%)	Speed (sents/sec)
CKY C&C	99.34	**86.79**	**96.3**
SR C&C	**99.58**	86.78	71.3
FP SR C&C	99.38	86.51	95.4
CKY C&C Auto	99.25	**84.59**	82.0
SR C&C Auto	**99.50**	84.53	61.2
FP SR C&C Auto	99.29	84.29	**84.9**

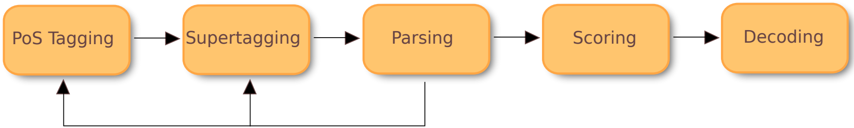
Table: Final evaluation on Section 23 of CCGbank

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 I

Stephen Clark and James R. Curran. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4):493–552, 2007.

Hany Hassan, Khalil Sima'an, and Andy Way. A Syntactic Language Model Based on Incremental CCG Parsing. In *Proceedings of the Workshop on Spoken Language Technology (SLT-08)*, pages 205–208, 2008.

Julia Hockenmaier and Mark Steedman. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.

References II

Tadao Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.

Jonathan K. Kummerfeld, Jessika Roesner, Tim Dawborn, James Haggerty, James R. Curran, and Stephen Clark. Faster Parsing by Supertagger Adaptation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL-10)*, pages 345–355, Uppsala, Sweden, July 2010.

Martin J. Pickering. Sentence comprehension. *Language Processing*, 2:123, 1999.

Mark Steedman. *The Syntactic Process*. MIT Press, Cambridge, Massachusetts, USA, 2000.

References III

Michael K. Tanenhaus and Sarah Brown-Schmidt. Language Processing in the Natural World. *Philosophical Transactions of the Royal Society of London*, 363(1493):1105–1122, 2008.

Masaru Tomita. Graph-structured Stack and Natural Language Parsing. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 249–257, Buffalo, New York, USA, June 1988.

Daniel H. Younger. Recognition and Parsing of Context-Free Languages in Time n^3. *Information and Control*, 10(2):189–208, February 1967.

References IV

Yue Zhang and Stephen Clark. Shift-Reduce CCG Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-11:HLT)*, pages 683–692, Portland, Oregon, USA, June 2011.