Chart Pruning for Fast Lexicalised-Grammar Parsing

Yue Zhang[a*] **Byung-Gyu Ahn**[b*] **Stephen Clark**[a*] **Curt Van Wyk**[c]
James R. Curran[d] **Laura Rimell**[a]

Computer Laboratory[a] Cambridge
Computer Science[b] Johns Hopkins
Computer Science[c] Northwestern College
School of IT[d] Sydney

{yue.zhang,stephen.clark}@cl.cam.ac.uk[a*] bahn@jhu.edu[b*]

Abstract

Given the increasing need to process massive amounts of textual data, efficiency of NLP tools is becoming a pressing concern. Parsers based on lexicalised grammar formalisms, such as TAG and CCG, can be made more efficient using supertagging, which for CCG is so effective that every derivation consistent with the supertagger output can be stored in a packed chart. However, wide-coverage CCG parsers still produce a very large number of derivations for typical newspaper or Wikipedia sentences. In this paper we investigate two forms of chart pruning, and develop a novel method for pruning complete cells in a parse chart. The result is a wide-coverage CCG parser that can process almost 100 sentences per second, with little or no loss in accuracy over the baseline with no pruning.

1 Introduction

Many NLP tasks and applications require the processing of massive amounts of textual data. For example, knowledge acquisition efforts can involve processing billions of words of text (Curran, 2004). Also, the increasing need to process large amounts of web data places an efficiency demand on existing NLP tools. TextRunner, for example, is a system that performs open information extraction on the web (Lin et al., 2009). However, the text processing that is performed by TextRunner, in particular the parsing, is rudimentary: finite-state shallow parsing technology that is now decades old. TextRunner uses this technology largely for efficiency reasons.

Many of the popular wide-coverage parsers available today operate at around one newspaper sentence per second (Collins, 1999; Charniak, 2000; Petrov and Klein, 2007). There are dependency parsers that operate orders of magnitude faster, by exploiting the fact that accurate dependency parsing can be achieved by using a shift-reduce linear-time process which makes a single decision at each point in the parsing process (Nivre and Scholz, 2004).

In this paper we focus on the Combinatory Categorial Grammar (CCG) parser of Clark and Curran (2007). One advantage of the CCG parser is that it is able to assign rich structural descriptions to sentences, from a variety of representations, e.g. CCG derivations, CCG dependency structures, grammatical relations (Carroll et al., 1998), and first-order logical forms (Bos et al., 2004). One of the properties of the grammar formalism is that it is lexicalised, associating CCG lexical categories, or CCG *supertags*, with the words in a sentence (Steedman, 2000). Clark and Curran (2004) adapt the technique of supertagging (Bangalore and Joshi, 1999) to CCG, using a standard maximum entropy tagger to assign small sets of supertags to each word. The reduction in ambiguity resulting from the supertagging stage results in a surprisingly efficient parser, given the rich structural output, operating at tens of newspaper sentences per second.

In this paper we demonstrate that the CCG parser can be made more than twice as fast, with little or no loss in accuracy. A noteworthy feature of the CCG parser is that, after the supertagging stage, the parser builds a complete packed chart, storing all sentences consistent with the assigned supertags and the parser's CCG combinatory rules, *with no chart pruning whatsoever*. The use of chart pruning techniques, typically some form of beam search, is essential for practical parsing using Penn Treebank parsers (Collins, 1999; Petrov and Klein, 2007; Charniak and Johnson, 2005), as well as practical parsers based on linguistic formalisms, such as HPSG (Ninomiya et al., 2005) and LFG (Kaplan et al., 2004). However, in the CCG case, the use of the supertagger means that enough ambiguity has already been resolved to allow the complete chart to be represented.

Despite the effectiveness of the supertagging stage, the number of derivations stored in a packed chart can still be enormous for typical newspaper sentences. Hence it is an obvious question whether chart pruning techniques can be profitably applied to the CCG parser. Some previous work (Djordjevic et al., 2007) has investigated this question but with little success.

In this paper we investigate two types of chart pruning: a standard beam search, similar to that used in the Collins parser (Collins, 1999), and a more aggressive strategy in which complete cells are pruned, following Roark and Hollingshead (2009). Roark and Hollingshead use a finite-state tagger to decide which words in a sentence can end or begin constituents, from which whole cells in the chart can be removed. We develop a novel extension to this approach, in which a tagger is trained to infer the maximum length constituent that can begin or end at a particular word. These lengths can then be used in a more agressive pruning strategy which we show to be significantly more effective than the basic approach.

Both beam search and cell pruning are highly effective, with the resulting CCG parser able to process almost 100 sentences per second using a single CPU, for both newspaper and Wikipedia data, with little or no loss in accuracy.

2 The CCG Parser

The parser is described in detail in Clark and Curran (2007). It is based on CCGbank, a CCG version of the Penn Treebank developed by Hockenmaier and Steedman (2007).

The stages in the parsing pipeline are as follows. First, a POS tagger assigns a single POS tag to each word in a sentence. Second, a CCG supertagger assigns lexical categories to the words in the sentence. Third, the parsing stage combines the categories, using CCG's combinatory rules, and builds a packed chart representation containing all the derivations which can be built from the lexical categories. Finally, the Viterbi algorithm finds the highest scoring derivation from the packed chart, using the normal-form log-linear model described in Clark and Curran (2007).

Sometimes the parser is unable to build an analysis which spans the whole sentence. When this happens the parser and supertagger interact using the adaptive supertagging strategy described in Clark and Curran (2004): the parser effectively asks the supertagger to provide more lexical categories for each word. This potentially continues for a number of iterations until the parser does create a spanning analysis, or else it gives up and moves to the next sentence.

The parser uses the CKY algorithm (Kasami, 1965; Younger, 1967) described in Steedman (2000) to create a packed chart. The CKY algorithm applies naturally to CCG since the grammar is binary. It builds the chart bottom-up, starting with two-word constituents (assuming the supertagging phase has been completed), incrementally increasing the span until the whole sentence is covered. The chart is packed in the standard sense that any two equivalent constituents created during the parsing process are placed in the same equivalence class, with pointers to the children used in the creation. Equivalence is defined in terms of the category and head of the constituent, to enable the Viterbi algorithm to efficiently find the highest scoring derivation.[1] A textbook treatment of CKY applied to statistical parsing is given in Jurafsky and Martin (2000).

3 Data and Evaluation Metrics

We performed efficiency and accuracy tests on newspaper and Wikipedia data. For the newspaper data, we used the standard test sections from

[1]Use of the Viterbi algorithm in this way requires the features in the parser model to be local to a single rule application; Clark and Curran (2007) has more discussion.

```
(ncmod num hundred_1 Seven_0)
(conj and_2 sixty-one_3)
(conj and_2 hundred_1)
(dobj in_6 total_7)
(ncmod _ made_5 in_6)
(aux made_5 were_4)
(ncsubj made_5 and_2 obj)
(passive made_5)

Seven hundred and sixty-one were made in
total.
```

Figure 1: Example Wikipedia test sentence annotated with grammatical relations.

CCGbank. Following Clark and Curran (2007) we used the CCG dependencies for accuracy evaluation, comparing those output by the parser with the gold-standard dependencies in CCGbank. Unlike Clark and Curran, we calculated recall scores over all sentences, including those for which the parser did not find an analysis. For the WSJ data the parser fails on a small number of sentences (less than 1%), but the chart pruning has the effect of reducing this failure rate further, and we felt that this should be factored into the calculation of recall and hence F-score.

In order to test the parser on Wikipedia text, we created two test sets. The first, Wiki 300, for testing accuracy, consists of 300 sentences manually annotated with grammatical relations (GRs) in the style of Briscoe and Carroll (2006). An example sentence is given in Figure 1. The data was created by manually correcting the output of the parser on these sentences, with the annotation being performed by Clark and Rimell, including checks on a subset of these cases to ensure consistency across the two annotators. For the accuracy evaluation, we calculated precision, recall and balanced F-measure over the GRs in the standard way.

For testing speed on Wikipedia, we used a corpus of 2500 randomly chosen sentences, Wiki 2500. For all speed tests we measured the number of sentences per second, using a single CPU and standard hardware.

4 Beam Search

The beam search approach used in our experiments prunes all constituents in a cell having scores below a multiple (β) of the score of the highest scoring constituent for that cell.[2] The scores for a constituent are calculated using the same model used to find the highest scoring derivation. We consider two scores: the Viterbi score, which is the score of the highest scoring sub-derivation for that constituent; and the inside score, which is the sum over all sub-derviations for that constituent. We investigated the following: the trade-off between the aggressiveness of the beam search and accuracy; the comparison between the Viterbi and inside scores; and whether applying the beam to only certain cells in the chart can improve performance.

β	Speed	Gain	F-score	Gain
Baseline	43.0		85.55	
0.001	48.6	13%	85.82	0.27
0.002	54.2	26%	85.88	0.33
0.005	59.0	37%	85.73	0.18
0.01	66.7	55%	85.53	-0.02

Table 1: Accuracy and speed results using different beam values β.

δ	Speed	Gain	F-score	Gain
Baseline	43.0		85.55	
10	60.1	39%	85.55	0.00
20	70.6	64%	85.66	0.11
30	72.3	68%	85.65	0.10
40	76.4	77%	85.63	0.08
50	76.7	78%	85.62	0.07
60	74.5	73%	85.71	0.16
80	68.4	59%	85.71	0.16
100	62.0	44%	85.73	0.18
None	59.0	37%	85.73	0.18

Table 2: Accuracy and speed results for different values of δ where $\beta = 0.005$.

Table 1 shows results on Section 00 of CCGbank, using the Viterbi score to prune. As expected, the parsing speed increases as the value of β increases, since more constituents are pruned with a higher β value. The pruning is effective, with a β value of 0.01 giving a 55% speed increase with neglible loss in accuracy.[3]

[2]One restriction we apply in practice is that only constituents resulting from the application of a CCG binary rule, rather than a unary rule, are pruned.

[3]The small accuracy increase for some β values could be attributable to two factors: one, the parser may select a lower

	Speed			F-score		
Dataset	Baseline	Beam	Gain	Baseline	Beam	Gain
WSJ 00	43.0	76.4	77%	85.55	85.63	0.08
WSJ 02-21	53.4	99.4	86%	93.61	93.27	-0.34
WSJ 23	55.0	107.0	94%	87.12	86.90	-0.22
Wiki 300	35.5	80.3	126%	84.23	85.06	0.83
Wiki 2500	47.6	90.3	89%			

Table 4: Beam search results on WSJ 00, 02-21, 23 and Wikipedia texts with $\beta = 0.005$ and $\delta = 40$.

	β	δ	Speed	F-score
Baseline			24.7	85.55
inside scores	0.01		37.7	85.52
	0.001		25.3	85.79
	0.005	10	33.4	85.54
	0.005	20	39.5	85.64
	0.005	50	42.9	85.58
Viterbi scores	0.01		38.1	85.53
	0.001		28.2	85.82
	0.005	10	33.6	85.55
	0.005	20	39.4	85.66
	0.005	50	43.1	85.62

Table 3: Comparison between using Viterbi scores and inside scores as beam scores.

We also studied the effect of the beam search at different levels of the chart. We applied a selective beam in which pruning is only applied to constituents of length less than or equal to a threshold δ. For example, if $\delta = 20$, pruning is applied only to constituents spanning 20 words or less. The results are shown in Table 2. The selective beam is also highly effective, showing speed gains over the baseline (which does not use a beam) with no loss in F-score. For a δ value of 50 the speed increase is 78% with no loss in accuracy.

Note that for δ greater than 50, the speed reduces. We believe that this is due to the cost of calculating the beam scores and the reduced effectiveness of pruning for cells with longer spans (since pruning shorter constituents early in the chart-parsing process prevents the creation of many larger, low-scoring constituents later).

Table 3 shows the comparison between the inside and Viterbi scores. The results are similar, with Viterbi marginally outperforming the inside score in most cases. The interesting result from these experiments is that the summing used in calculating the inside score does not improve performance over the max operator used by Viterbi.

Table 4 gives results on Wikipedia text, compared with a number of sections from CCGbank. (Sections 02-21 provide the training data for the parser which explains the high accuracy results on these sections.) Despite the fact that the pruning model is derived from CCGbank and based on WSJ text, the speed improvements for Wikipedia were even greater than for WSJ text, with parameters $\beta = 0.005$ and $\delta = 40$ leading to almost a doubling of speed on the Wiki 2500 set, with the parser operating at 90 sentences per second.

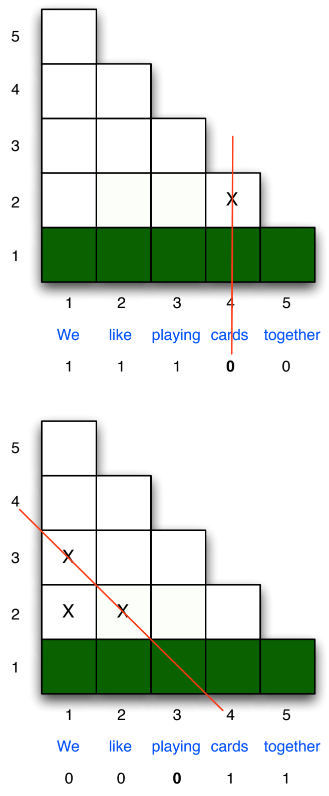
5 Cell Pruning

Whole cells can be pruned from the chart by tagging words in a sentence. Roark and Hollingshead (2009) used a binary tagging approach to prune a CFG CKY chart, where tags are assigned to input words to indicate whether they can be the start or end of multiple-word constituents. We adapt their method to CCG chart pruning. We also show the limitation of binary tagging, and propose a novel tagging method which leads to increased speeds and accuracies over the binary taggers.

5.1 Binary tagging

Following Roark and Hollingshead (2009), we assign the binary begin and end tags separately using two independent taggers. Given the input "We like playing cards together", the pruning effects of each type of tag on the CKY chart are shown in Figure 2. In this chart, rows repre-

scoring but more accurate derivation; and two, a possible increase in recall, discussed in Section 3, can lead to a higher F-score.



5
4
3
2
1
X
1 2 3 4 5
We like playing cards together
1 1 1 0 0
5
4
3
2
1
X
X
X
1 2 3 4 5
We like playing cards together
0 0 0 1 1

Figure 2: The pruning effect of begin (top) and end (bottom) tags; X indicates a removed cell.

sent consituent sizes and columns represent initial words of constituents. No cell in the first row of the chart is pruned, since these cells correspond to single words, and are necessary for finding a parse. The begin tag for the input word "cards" is 0, which means that it cannot begin a multi-word constituent. Therefore, no cell in column 4 can contain any constituent. The pruning effect of a binary begin tag is to cross out a column of chart cells (ignoring the first row) when the tag value is zero. Similarly, the end tag of the word "playing" is 0, which means that it cannot be the end of a multi-word constituent. Consequently cell (2, 2), which contains constituents for "like playing", and cell (1, 3), which contains constituents for "We like playing", must be empty. The pruning effect of a binary end tag is to cross out a diagonal of cells (ignoring the first row) when the tag value is zero.

We use a maximum entropy trigram tagger (Ratnaparkhi, 1996; Curran and Clark, 2003) to assign the begin and end tags. Features based on the words and POS in a 5-word window, plus the two previously assigned tags, are extracted from the trigram ending with the current tag and the five-word window with the current word in the middle. In our development experiments, both the begin and the end taggers gave a per-word accuracy of around 96%, similar to the accuracy reported in Roark and Hollingshead (2009).

Model	Speed	F-score
baseline	25.10	84.89
begin only	27.49	84.71
end only	30.33	84.56
both	33.90	84.60
oracle	33.60	85.67

Table 5: Accuracy and speed results for the binary taggers on Section 00 of CCGbank.

Table 5 shows accuracy and speed results for the binary taggers.[4] Using begin or end tags alone, the parser achieved speed increases with a small loss in accuracy. When both begin and end tags are applied, the parser achieved further speed increases, with no loss in accuracy compared to the end tag alone. Row "oracle" shows what happens using the perfect begin and end taggers, by using gold-standard constituent information from CCGbank. The F-score is higher, since the parser is being guided away from incorrect derivations, although the speed is no higher than when using automatically assigned tags.

5.2 Level tagging

A binary tag cannot take effect when there is any chart cell in the corresponding column or diagonal that contains constituents. For example, the begin tag for the word "card" in Figure 3 cannot be 0 because "card" begins a two-word constituent "card games". Hence none of the cells in the column can be pruned using the binary begin tag, even though all the cells from the third row above are empty. We propose what we call a level tagging approach to address this problem.

Instead of taking a binary value that indicates

[4]The baseline differs slightly to the previous section because gold-standard POS tags were used for the beam-search experiments.

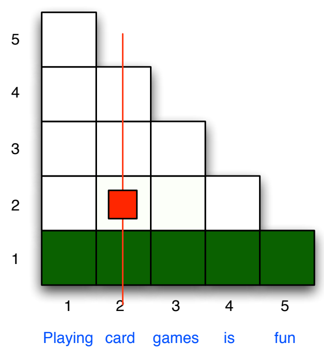


Figure 3: The limitation of binary begin tags.

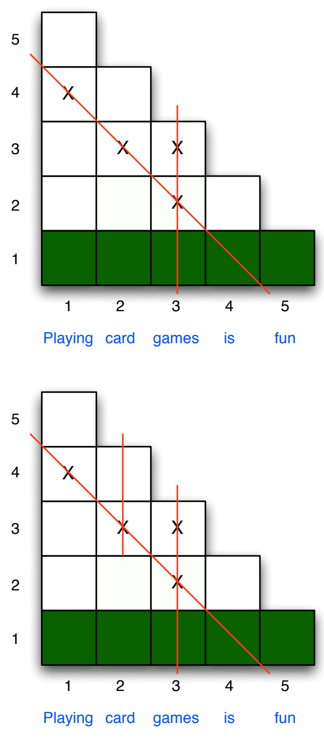


Figure 4: The pruning effect of binary (top) and level (bottom) tags.

whether a whole column or diagonal of cells can be pruned, a level tag (begin or end) takes an integer value which indicates the row from which a column or diagonal can be pruned in the upward direction. For example, a level begin tag with value 2 allows the column of chart cells for the word "card" in Figure 3 to be pruned from the third row upwards. A level tag (begin or end) with value 1 prunes the corresponding row or diagonal from the second row upwards; it has the same pruning effect as a binary tag with value 0. For convenience, value 0 for a level tag means that the corresponding word can be the beginning or end of any constituent, which is the same as a binary tag value 1.

A comparison of the pruning effect of binary and level tags for the sentence "Playing card games is fun" is shown in Figure 4. With a level begin tag, more cells can be pruned from the column for "card". Therefore, level tags are potentially more powerful for pruning.

We now need a method for assigning level tags to words in a sentence. However, we cannot achieve this with a straighforward classifier since level tags are related; for example, a level tag (begin or end) with value 2 implies level tags with values 3 and above. We develop a novel method for calculating the probability of a level tag for a particular word. Our mechanism for calculating these probabilities uses what we call *maxspan* tags, which can be assigned using a maximum entropy tagger.

Maxspan tags take the same values as level tags. However, the meanings of maxspan tags and level tags are different. While a level tag indicates the row from which a column or diagonal of cells is pruned, a maxspan tag represents the size of the largest constituent a word begins or ends. For example, in Figure 3, the level end tag for the word "games" has value 3, since the largest constituent this words ends spans "playing card games".

We use the standard maximum entropy trigram tagger for maxspan tagging, where features are extracted from tag trigrams and surrounding five-word windows, as for the binary taggers. Parse trees can be turned directly into training data for a maxspan tagger. Since the level tag set is finite, we a require a maximum value N that a level tag can take. We experimented with $N = 2$ and $N = 4$, which reflects the limited range of the features used by the taggers.[5]

During decoding, the maxspan tagger uses the forward-backward algorithm to compute the probability of maxspan tag values for each word in the

[5]Higher values of N did not lead to improvements during development experiments.

Model	Speed	F-score
baseline	25.10	84.89
binary	33.90	84.60
binary oracle	33.60	85.67
level $N = 2$	32.79	84.92
level $N = 4$	34.91	84.95
level $N = 4$ oracle	47.45	86.49

Table 6: Accuracy and speed results for the level taggers on Section 00 of CCGbank.

input. Then for each word, the probability of its level tag t_l having value x is the sum of the probabilities of its maxspan t_m tag having values $1..x$:

$$P(t_l = x) = \sum_{i=1}^{x} P(t_m = i)$$

Maxspan tag values i from 1 to x represent disjoint events in which the largest constituent that the corresponding word begins or ends has size i. Summing the probabilities of these disjoint events gives the probability that the largest constituent the word begins or ends has a size between 1 and x, inclusive. That is also the probability that all the constituents the word begins or ends are in the range of cells from rows 1 to row x in the corresponding column or diagonal. And therefore that is also the probability that the chart cells above row x in the corresponding column or diagonal do not contain any constituents, which means that the column and diagonal can be pruned from row x upward. Therefore, it is also the probability of a level tag with value x.

The probability of a level tag having value x increases as x increases from 1 to N. We set a probability threshold Q and choose the smallest level tag value x with probability $P(t_l = x) \geq Q$ as the level tag for a word. If $P(t_l = N) < Q$, we set the level tag to 0 and do not prune the column or diagonal. The threshold value determines a balance between pruning power and accuracy, with a higher value pruning more cells but increasing the risk of incorrectly pruning a cell. During development we arrived at a threshold value of 0.8 as providing a suitable compromise between pruning power and accuracy.

Table 6 shows accuracy and speed results for the level tagger, using a threshold value of 0.8.

Model	Speed	F-score
baseline	36.64	84.23
binary gold	49.59	84.36
binary self 40K	48.79	83.64
binary self 200K	51.51	83.71
binary self 1M	47.78	83.75
level gold	58.23	84.12
level self 40K	54.76	83.83
level self 200K	48.57	83.39
level self 1M	52.54	83.71

Table 7: Accuracy tests on Wiki 300 comparing gold training (gold) with self training (self) for different sizes of parser output for self-training.

We compare the effect of the binary tagger and level taggers with $N = 2$ and $N = 4$. The accuracies with the level taggers are higher than those with the binary tagger; they are also higher than the baseline parsing accuracy. The parser achieves the highest speed and accuracy when pruned with the $N = 4$ level tagger. Comparing the oracle scores, the level taggers lead to higher speeds than the binary tagger, reflecting the increased pruning power of the level taggers compared with the binary taggers.

5.2.1 Final experiments using gold training and self training

In this section we report our final tests using Wikipedia data. We used two methods to derive training data for the taggers. The first is the standard method, which is to transform gold-standard parse trees into begin and end tag sequences. This method is the method that we used for all previous experiments, and we call it “gold training”. In addition to gold training, we also investigate an alternative method, which is to obtain training data for the taggers from the output of the parser itself, in a form of self-training (McClosky et al., 2006). The intuition is that the tagger will learn what constituents a trained parser will eventually choose, and as long as the constituents favoured by the parsing model are not pruned, no reduction in accuracy can occur. There is the potential for an increase in speed, however, due to the pruning effect.

For gold training, we used sections 02-21 of

Model	Speed
baseline	47.6
binary gold	80.8
binary 40K	75.5
binary 200K	77.4
binary 1M	78.6
level gold	93.7
level 40K	92.8
level 200K	92.5
level 1M	96.6

Table 8: Speed tests with gold and self-training on Wiki 2500.

CCGBank (which consists of about 40K training sentences) to derive training data. For self training, we trained the parser on sections 02-21 of CCGBank, and used the parser to parse 40 thousand, 200 thousand and 1 million sentences from Wikipedia, respectively. Then we derive three sets of self training data from the three sets of parser outputs. We then used our Wiki 300 set to test the accuracy, and the Wiki 2500 set to test the speed of the parser.

The results are shown in Tables 7 and 8, where each row represents a training data set. Rows "binary gold" and "level gold" represent binary and level taggers trained using gold training. Rows "binary self X" and "level self X" represent binary and level taggers trained using self training, with the size of the training data being X sentences.

It can be seen from the Tables that the accuracy loss with self-trained binary or level taggers was not large (in the worst case, the accuracy dropped from 84.23% to 83.39%), while the speed was significantly improved. Using binary taggers, the largest speed improvement was from 47.6 sentences per second to 80.8 sentences per second (a 69.7% relative increase). Using level taggers, the largest speed improvement was from 47.6 sentences per second to 96.6 sentences per second (a 103% relative increase).

A potential advantage of self-training is the availability of large amounts of training data. However, our results are somewhat negative in this regard, in that we find training the tagger on more than 40,000 parsed sentences (the size of CCGbank) did not improve the self-training results. We did see the usual speed improvements from using the self-trained taggers, however, over the baseline parser with no pruning.

6 Conclusion

Using our novel method of level tagging for pruning complete cells in a CKY chart, the CCG parser was able to process almost 100 Wikipedia sentences per second, using both CCGbank and the output of the parser to train the taggers, with little or no loss in accuracy. This was a 103% increase over the baseline with no pruning.

We also demonstrated that standard beam search is highly effective in increasing the speed of the CCG parser, despite the fact that the supertagger has already had a significant pruning effect. In future work we plan to investigate the gains that can be achieved from combining the two pruning methods, as well as other pruning methods such as the self-training technique described in Kummerfeld et al. (2010) which reduces the number of lexical categories assigned by the supertagger (leading to a speed increase). Since these methods are largely orthogonal, we expect to achieve further gains, leading to a remarkably fast wide-coverage parser outputting complex linguistic representations.

Acknowledgements

This work was largely carried out at the Johns Hopkins University Summer Workshop and (partially) supported by National Science Foundation Grant Number IIS-0833652. Yue Zhang and Stephen Clark are supported by the European Union Seventh Framework Programme (FP7-ICT-2009-4) under grant agreement no. 247762. James Curran is supported by Australian Research Council Discovery grants DP0665973 and DP1097291 and the Capital Markets Cooperative Research Centre.

References

Bangalore, Srinivas and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Bos, Johan, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of COLING-04*, pages 1240–1246, Geneva, Switzerland.

Briscoe, Ted and John Carroll. 2006. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of the Poster Session of COLING/ACL-06*, pages 41–48, Sydney, Australia.

Carroll, John, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st LREC Conference*, pages 447–454, Granada, Spain.

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine N-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Meeting of the ACL*, pages 173–180, Michigan, Ann Arbor.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the NAACL*, pages 132–139, Seattle, WA.

Clark, Stephen and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING-04*, pages 282–288, Geneva, Switzerland.

Clark, Stephen and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Collins, Michael. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Curran, James R. and Stephen Clark. 2003. Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 10th Meeting of the EACL*, pages 91–98, Budapest, Hungary.

Curran, James R. 2004. *From Distributional to Semantic Similarity*. Ph.D. thesis, University of Edinburgh.

Djordjevic, Bojan, James R. Curran, and Stephen Clark. 2007. Improving the efficiency of a wide-coverage CCG parser. In *Proceedings of IWPT-07*, pages 39–47, Prague, Czech Republic.

Hockenmaier, Julia and Mark Steedman. 2007. CCG-bank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Jurafsky, Daniel and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, New Jersey.

Kaplan, Ron, Stefan Riezler, Tracy H. King, John T. Maxwell III, Alexander Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of HLT-NAACL'04*, Boston, MA.

Kummerfeld, Jonathan K., Jessika Roesner, Tim Dawborn, James Haggerty, James R. Curran, and Stephen Clark. 2010. Faster parsing by supertagger adaptation. In *Proceedings of ACL-10*, Uppsala, Sweden.

Lin, Thomas, Oren Etzioni, and James Fogarty. 2009. Identifying interesting assertions from the web. In *Proceedings of the 18th Conference on Information and Knowledge Management (CIKM 2009)*, Hong Kong.

McClosky, David, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of NAACL-06*, pages 152–159, Brooklyn, NY.

Ninomiya, Takashi, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic HPSG parsing. In *Proceedings of IWPT-05*, pages 103–114, Vancouver, Canada.

Nivre, J. and M. Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of COLING-04*, pages 64–70, Geneva, Switzerland.

Petrov, Slav and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of the HLT/NAACL conference*, Rochester, NY.

Ratnaparkhi, Adwait. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP-96*, pages 133–142, Somerset, New Jersey.

Roark, Brian and Kristy Hollingshead. 2009. Linear complexity context-free parsing pipelines via chart constraints. In *Proceedings of HLT/NAACL-09*, pages 647–655, Boulder, Colorado.

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