Punctuation normalisation for cleaner treebanks and parsers

Daniel Tse and **James R. Curran**
School of Information Technologies
University of Sydney
NSW 2006, Australia
{dtse6695,james}@it.usyd.edu.au

Abstract

Although punctuation is pervasive in written text, their treatment in parsers and corpora is often second-class.

We examine the treatment of commas in CCGbank, a wide-coverage corpus for Combinatory Categorial Grammar (CCG), reanalysing its comma structures in order to eliminate a class of redundant rules, obtaining a more consistent treebank.

We then eliminate these rules from C&C, a wide-coverage statistical CCG parser, obtaining a 37% increase in parsing speed on the standard CCGbank test set and a considerable reduction in memory consumed, without affecting parser accuracy.

1 Introduction

Although parsers must all at least accept text containing punctuation, there is much variation in how punctuation is treated during training, parsing and evaluation. The work of Nunberg (1990) brought attention to the fact that parsers can often reject otherwise ambiguous parses on the basis of punctuation, by considering a formal grammar of text units: chunks of text delimited by punctuation tokens.

In this work, we explore how the treatment of commas, the most common class of punctuation in the Penn Treebank (Marcus et al., 1994), affects their analysis in CCGbank (Hockenmaier and Steedman, 2007), a 1.2 million word corpus for Combinatory Categorial Grammar (CCG), generated from a subset of the Penn Treebank. We uncover a systematic inconsistency in the way CCGbank represents sentences involving commas, and perform a transformation of the original corpus which eliminates this source of uninformative variation, resulting in a version of CCGbank which assigns a uniform structure to each of the syntactic roles played by commas in the corpus.

By removing the superfluous rules induced by this inconsistency from the corpus, we obtain cleaner analyses of comma structures which require fewer CCG rules to explain. These changes to the corpus, in turn, allow us to simplify the implementation of the C&C parser (Clark and Curran, 2007), a wide-coverage statistical CCG parser for English, by barring the parser from considering the CCG rules which we have made redundant.

Training the C&C parser on our modified CCGbank yields average speed gains of 37% and 21% on the standard CCGbank test set and development set respectively, and a 47% reduction in memory consumed by the chart parsing process as evaluated on the standard CCGbank test set. Parser coverage increases slightly while accuracy is not affected.

Consistency, a powerful and general guideline in corpus design, can improve the usefulness and quality of corpora and the applications built with them. We eliminate a source of systematic inconsistency in comma representation from a wide-coverage corpus and parser, resulting in cleaner versions of both resources. The combination of a cleaner treebank and parser leads to gains in speed and a reduction in memory consumption without affecting parser accuracy.

2 Background

Nunberg (1990) characterises the traditional view of written language as a simulacrum of a richer medium, that of spoken language. Nunberg notes that although formal prescriptive accounts of punctuation have long been the domain of grammarians (and the bane of their students), this perception of writing as an inferior or merely transcriptional medium has lead to the relative paucity of analytic accounts of punctuation in linguistics.

Nunberg (1990) describes the macro-structure of English text as two superposed systems: the *text grammar* defines the well-formedness of a sentence interpreted as a sequence of not necessarily constituent-forming units, while the *lexical grammar* encodes the usual relationships between individual lexical items.

Nunberg's validation of punctuation as a linguistic system meriting study in its own right spurred work affording punctuation a more integral role in generation and parsing (Briscoe, 1994; Osborne, 1995), as well as work in corpus linguistics as to the distribution and semantics of commas, and other classes of punctuation tokens (Bayraktar et al., 1998; Jones, 1997).

However, despite this renewal of interest, punctuation is still often relegated to a marginal role in parsing and the evaluation of parsers, because its representation is often inconsistent between treebanks, and even within a treebank. `evalb`, a widely used script for calculating the crossing-brackets parser evaluation metric PARSEVAL, strips all punctuation from the candidate and gold standard text (Sekine and Collins, 2006), so that the attachment level of any punctuation does not influence the bracketing metric.

Briscoe (1994) discusses a parsing system which uses Nunberg's concept of a text grammar to inform its chunking decisions (in this context, the segmentation of input text into sentences, robustly handling non-terminal uses of periods, for example). In Briscoe's work, an implementation of text grammar can eliminate incorrect parses based on punctuation. In Example 1, a variation on the well-known *PP* attachment problem, a parser which simply ignores punctuation in the input will encounter ambiguity which a punctuation-aware parser will not.

(1) I saw the girl on the hill with the telescope, from the store.

(2) I saw the girl on the hill with the telescope from the store.

The work of Djordjevic et al. (2007) for the C&C parser constrains phrases delimited by hyphens, colons and semicolons to be complete constituents, allowing the parser to avoid great amounts of attachment ambiguity when parsing long sentences. However, while Djordjevic et al. modify the parser to derive information from punctuation in the corpus, this work changes the representation of commas in the corpus to convey information on each comma's role to the parser. The corpus we obtain assigns a uniform structure to each of the comma's syntactic roles.

3 CCG and CCGbank

In Combinatory Categorial Grammar (Steedman, 2000), each lexical item receives a *category* such as N/N or $(S\backslash NP)/NP$, which determines how it may combine with other groups of lexical items. New categories may be formed of adjacent categories by applying *combinatory rules*. In Figure 1, each line denotes the application of a particular combinatory rule, which combines at most two categories into another category. Combinatory rules enable succinct, natural CCG analyses for difficult constructions such as argument cluster coordination and parasitic gaps (Steedman, 2000).

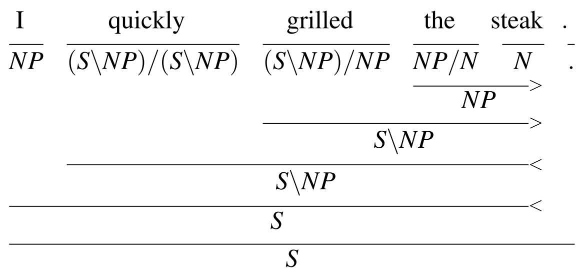

Figure 1: CCG analysis

CCGbank is a wide-coverage CCG corpus, generated from the Wall Street Journal section of Penn Treebank through an automatic conversion procedure described by Hockenmaier and Steedman (2007). CCGbank enabled the development of wide-coverage statistical CCG parsers such as the C&C parser Clark and Curran (2007), on which we will evaluate this work. Parsing in the C&C parser proceeds in two broad stages: a maximum entropy *supertagger* assigns categories to lexical items, which the *parser* then attempts to combine using the CCG rules. The corpus, as a collection of CCG derivations, implicitly encodes the set of CCG rules used in its derivations. These rules must be explicitly encoded in a parser trained on CCGbank, since the training process requires it to reproduce the derivations represented in the corpus. The goal of this work is to drastically reduce the number of rules implicitly encoded by CCGbank, and in turn reduce the number of rules explicitly encoded in the C&C parser.

The distribution of punctuation symbols in CCGbank is given in Figure 2. Commas account for over half of all punctuation tokens, and periods well-represented but not as common as commas. (Bayraktar et al., 1998) observes the same dramatic drop between the frequency of commas and periods, and the remaining punctuation marks.

Symbol	Frequency	
comma	59991	(53.77%)
period	47875	(42.91%)
colon	1649	(1.48%)
semi-colon	1460	(1.31%)
question mark	511	(0.46%)
excl. mark	71	(0.06%)
apostrophe	3	(0.002%)
Total punct	111560	

Figure 2: Punctuation distribution in CCGbank

Although periods approach commas in frequency, their representation in CCGbank is largely consistent, since all of their roles are constituent-final: sentence terminator, list index delimiter, abbreviation marker. By comparison, commas in CCGbank occur both constituent-initially as well as finally.

The goal of this work is to standardise the representation of commas in CCGbank, either by converting all *left commas* (comma leaves which are the left child of their parent) to *right commas*, or vice versa, and in doing so, allow us to remove unnecessary rules from the parser. In the next section, we discuss and justify which of the CCGbank rules are unnecessary, and which CCGbank rules our normalisation procedure should not modify.

4 Normalising CCGbank

Our goal of eliminating uninformative variation from the corpus requires us to distinguish cases where comma direction conveys some useful information from those where it has no discriminative function. We partition the CCGbank rules involving a comma argument so that our processing only affects those cases we are free to manipulate.

The intuition for our transformations on CCGbank are as follows: commas are by a large margin, the most common form of punctuation encountered in Penn Treebank (Bayraktar et al., 1998).

The inconsistency in the representation of commas is an artifact of the procedure by which Penn Treebank derivations are transformed into CCGbank derivations. Given a Penn Treebank derivation P, the procedure uses head-finding heuristics to identify the head of each constituent. Binarisation is a consequence of the fact that CCG's combinatory rules are all unary or binary, requiring us to transform the k-way branching structures of Penn Treebank-style annotation. The procedure uses the heads identified by the first step to determine whether to generate left- or right-branching structures: constituents left of the head are left-branching, and those right of the head branch rightwards, as depicted in Figure 3. This ensures that adjuncts seek the head, and not vice versa.

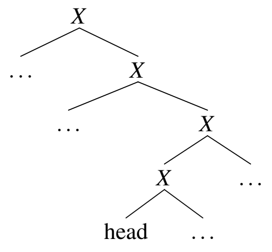

Figure 3: CCGbank binarisation

Accordingly, whether or not a comma node is a left, or a right comma, depends on whether it was left or right of the node identified by the head-finding heuristic as the head, as shown in Figure 4.

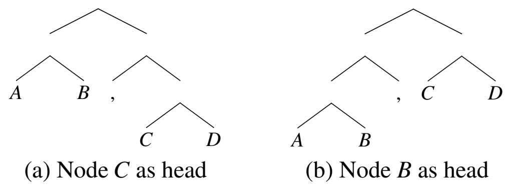

Figure 4: Left and right commas

This means that for many categories, the corpus contains a pair of *absorption rules*, so called because they leave the input category X unchanged:

$$\begin{array}{cccc} X & , & \rightarrow & X \\ , & X & \rightarrow & X \end{array}$$

which, by virtue of being extra-combinatory rules, must each be encoded in the parser. This proliferation of extra rules is merely an artifact of the binarisation process; in these cases, we judge that there is no evidence for the comma belonging to one side or the other, and that we are free to process the corpus so that it embodies only one of the above two rules.

4.1 Type-change rules

Type-change rules in CCGbank analyse syntax such as apposition without creating additional categorial ambiguity. The tradeoff is that such rules are completely unlicensed by the underlying theory, hence the designation *extra-combinatory*. Hockenmaier and Steedman (2007) argue for the use of extra-combinatory rules, to mitigate against the sparsity of data which would be entailed by giving every apposition *NP* a different category to an *NP* not participating in apposition. For example, the CCGbank analysis of sentence-final *NP* apposition:

(3) The index fell to 997, the first decline since July.

involves the following comma type-change rule:

$$, \quad NP \quad \rightarrow \quad (S\backslash NP)\backslash(S\backslash NP) \quad (\mathbf{T},)$$

The rule specifies that an *NP* following a comma can be treated as a verb phrase modifier (in general, the category of a modifier is X/X or $X\backslash X$ for some category X), allowing the analysis shown in Figure 5.

In CCGbank, the comma introducing an apposition is necessary to the resulting analysis, in that the rule does not trigger unless the comma is present. As such, comma type-change rules are fundamentally different from the comma absorption rules seen above, which neither introduce dependencies, nor affect interpretation. Furthermore, we can see that the above rule cannot involve anything but a left comma, for the simple reason that it is the analysis for a sentence-*final NP* apposition. Therefore, if we were to blindly normalise the above rule to:

$$^{*}NP \quad , \quad \rightarrow \quad (S\backslash NP)\backslash(S\backslash NP) \quad (\mathbf{T},)$$

we would no longer attain the same analysis of sentence-final *NP* apposition. In general, Bayraktar et al. (1998) consider the commas which introduce apposition to be a form of paired punctuation, like quotes or parentheses. Unlike close quotes, or close parentheses, however, the 'closing comma' is suppressed when it immediately precedes the sentence-final period. This suggests that it should be the 'opening comma' that triggers the use of type-change rules enabling apposition, since the 'closing comma' is not realised sentence-finally, as shown in Example 5.

(4) Pierre Vinken, chairman of Elsevier, will become non-executive director.

(5) The non-executive director is Pierre Vinken, chairman of Elsevier.

Hence, our comma normalisation does not process commas involved in comma type-change rules.

4.2 Conjunction commas

CCG is known for its straightforward analysis of coordination: two categories may be coordinated with a conjunction precisely when the conjuncts have the same category (Steedman, 2000). The CCGbank treatment of coordination differs from the rule of syncategorematic coordination originally given by Steedman (2000), because of the undesirability of introducing such a ternary rule as an exception to a system of otherwise unary and binary combinatory rules. To analyse coordination of nouns in series:

(6) Pound cake consists of butter, eggs and flour.

CCGbank instead provides the following pair of extra-combinatory rules:

$$\begin{array}{rll} , & N \rightarrow N[conj] & (\Phi_1) \\ N & N[conj] \rightarrow N & (\Phi_2) \end{array}$$

which yield an analysis isomorphic to Steedman's ternary coordination rule, without the need to directly implement such a ternary rule.

A category of the form $X[conj]$ represents a conjunct which has picked up a conjunction word, but not the other conjunct[1].

[1]In the analysis of Figure 6, a comma inserted before '*and*' (the so-called *Oxford comma*) would receive the comma category , and be subject to comma absorption, with the lexical conjunction '*and*' serving as the actual conjunction word.

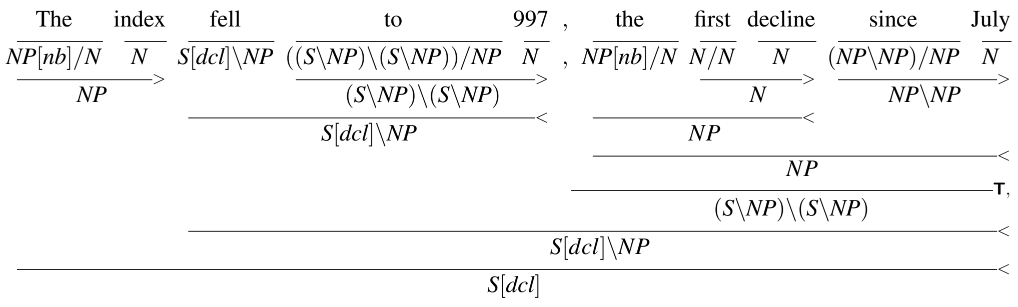
The index fell to 997 , the first decline since July
NP[nb]/N N S[dcl]\NP ((S\NP)\(S\NP))/NP N , NP[nb]/N N/N N (NP\NP)/NP N
NP
(S\NP)\(S\NP)
N
NP\NP
S[dcl]\NP
NP
NP
T,
(S\NP)\(S\NP)
S[dcl]\NP
S[dcl]

Figure 5: The type change rule $, \quad NP \rightarrow (S\backslash NP)\backslash(S\backslash NP)$ is applied in the fifth row

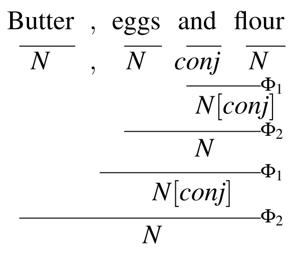

Figure 6: CCGbank analysis of coordination

For example, the partial coordination *and flour* in Figure 6 receives the category $NP[conj]$.

Suppose we are to convert the left comma rule Φ_1 to a right comma rule as a candidate for normalisation.

$$N \quad , \quad \rightarrow N[conj] \qquad (\Phi_1')$$

Such a rule has the effect of consuming conjuncts left-to-right, instead of right-to-left in the canonical set of rules. To support such an order, we would also need to change rule Φ_2 accordingly:

$$N[conj] \quad N \rightarrow N \qquad (\Phi_2')$$

That is, while the canonical set of rules yields a right-branching analysis of coordination, replacing the left comma rule with a right comma rule yields a left-branching analysis instead. Functionally, there is no difference between a left- and right-branching analysis. However, if we were to convert the left comma coordination rule Φ_1 to a right comma rule, we would have to re-analyse all right-branching coordination structures in the corpus as left-branching structures, with no change in semantics. Furthermore, we would save the parser no effort, since the original rule Φ_1 does not have a right comma rule analogue which we could eliminate. Therefore, we exclude coordination comma rules as candidates for comma normalisation.

4.3 Left, or right commas?

We now have a choice between normalising all absorption commas to left commas or to right commas. We have seen that the two non-absorption classes of comma rules both involve left commas, while absorption comma rules occur both as left and right comma rules. We make the arbitrary choice to normalise all absorption comma rules to right comma rules, so that it is easier for corpus applications to distinguish the most common case of punctuation absorption from other "special" rules such as type-changing and coordination.

Although an additional reason for normalising to right commas rather than left commas is not evident from our analysis of the *unmodified* CCGbank, an argument arises when we consider the version of CCGbank described by Tse and Curran (2007), which restores quote symbols to CCGbank. The re-quoting procedure operates by recovering the positions of open and close quotes from CCGbank's source corpus, Penn Treebank. If the text lying between the start and end of the quoted span exactly spans a subtree, then we can enclose that entire subtree in a quoting structure. Otherwise, we attach the open and close quotes separately. However, a common convention in English prose style is to include the comma which offsets a span of quoted direct speech as part of the span of quoted text.

(7) "I love hot dogs," said Tom with relish.

In the unmodified corpus, such a comma is attached to the right subtree, as in Figure 7a. However, this prevents us from enclosing the span of quoted text (including the comma), since the comma is stranded in the right-hand subtree. Accordingly, we must transform such a left comma to a right comma in order to obtain an analysis in which the open and close quotes correctly enclose the span of quoted text, as in Figure 7b.

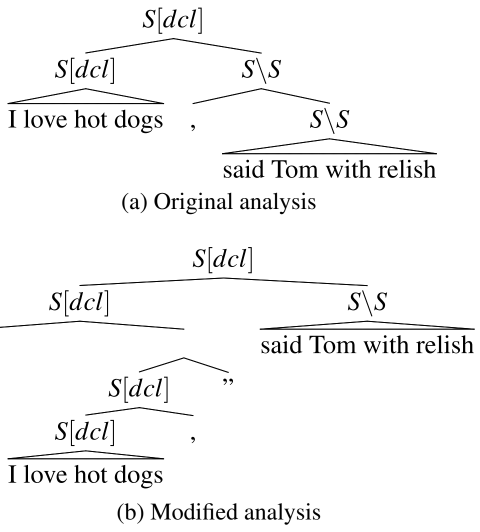

Figure 7: Quote insertion and comma direction

The fact that the correct analysis of such quoting structures *requires* right comma absorption suggests that for consistency, we should normalise to right, rather than left commas.

To summarise, our criteria for normalisation are to process only absorption commas, leaving type-change comma rule instances and analyses of coordination unmodified. The normalisation of all absorption commas to right commas, which enables us to remove all left comma absorption rules from the parser, allows us to make accuracy gains and speed reductions in the parser.

We also hypothesise that no useful information is lost as long as we restrict comma normalisation to the absorption cases we have identified above, since commas involved in absorption rules do not project dependencies, and hence cannot adversely affect the standard dependency-based CCGbank evaluation we describe in Section 7.

5 Transformations

Having shown that we are free to manipulate instances of absorption comma rules, the transformations themselves are simple. The procedure for comma normalisation is given in Algorithm 1.

We choose to re-attach the comma node as high as possible in the derivation tree, as shown in Figure 8b, in the absence of a more sophisticated method of determining the depth in the left subtree at which the comma should be restored. With reference to Figure 8, the algorithm has converted every instance of a left absorption rule $(, \quad Y \longrightarrow Y)$ to an instance of some right absorption rule $(X \quad , \longrightarrow X)$. Performing this transformation on every absorption comma in the corpus achieves the desired normalisation.

Algorithm 1 NORMALISE-COMMAS(C):

```
for each leaf $l$ in $C$ do
  if $(l, l.sibling \rightarrow l.parent)$ is comma absorption then
    $cur \leftarrow l$.parent
    Delete $l$ and its sibling
    {Re-attach the comma as high as possible}
    while $cur$ is the left child of its parent do
      $cur \leftarrow cur$.parent
    end while
    Insert comma structure at sibling of $cur$
  end if
end for
```

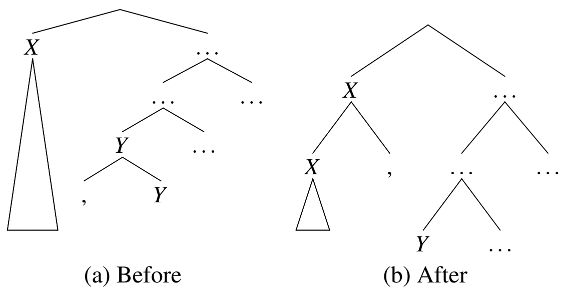

Figure 8: Comma movement

Figure 9 shows the distribution of occurrences of each kind of comma rule before, and after comma normalisation. The frequencies for two CCG rules for coordination which differ in whether a comma, or a lexical conjunction is used as the coordinating word are aggregated in the second row.

Our procedure successfully re-analyses every left absorption comma in CCGbank as an instance of a right absorption rule. The two instances of left commas which remain in the comma normalised corpus result from mis-tokenisations in Penn Treebank: a numeral and a determiner mis-tagged as a comma.

As a final step, we remove support for the now-eliminated left comma rules from the C&C parser,

CCG rule	Before	After
$, \quad X \quad \rightarrow X$	25690	2
$X \quad , \quad \rightarrow X$	21990	47678
$\{,\|conj\} \quad X \quad \rightarrow X[conj]$	11466	11466
Other typechange	607	607
$, \quad NP \rightarrow (S\backslash NP)\backslash(S\backslash NP)$	242	242
Total	*59995*	*59995*

Figure 9: Number of comma rule instances before and after normalisation

preventing the parser from hypothesising the attachment of commas to the left in the absorption case, reserving left-attachment analyses for the special cases of coordination and type-change structures.

6 Evaluation

We wish to determine the impact of our changes in two respects: their effect on the accuracy of the resulting parser model and on parser ambiguity, relative to the unmodified parser and corpus.

To measure the impact of the modifications on parser accuracy, we performed the dependency-based evaluation of Clark and Curran (2004) on the standard development and test sets, sections 00 and 23 of CCGbank. We evaluate the performance of a parser and model by computing the F-score over the obtained set of predicate-argument dependencies, relative to a gold standard. In the *labelled* CCG evaluation, a dependency consists of a tuple $\langle H, C, i, D, l \rangle$, where H is the head lexical item, C is its category with its arguments annotated with indices, i is the index of the argument satisfied by this dependency, and D is the lexical item satisfying this dependency. l is a flag distinguishing local from long-range dependencies. The *unlabelled* evaluation is performed against reduced tuples $\langle H, D \rangle$. Additional evaluation metrics are *sentence level accuracy*: the proportion of sentences for which the parser produced every gold standard dependency, *category accuracy*: the proportion of correctly assigned categories, and *coverage*: the proportion of sentences for which the parser obtained some spanning analysis. The column *Auto F* denotes the labelled F score when the C&C parser assigns its own POS tags instead of using gold standard tags.

To gauge the impact on parser ambiguity and memory consumption, we consider the number of *conjunctive nodes* generated by the C&C parser during the parsing process. The C&C parser uses a *packed chart representation* to reduce the size in memory of the parsing chart by allowing partial derivations which span the same category and have the same set of unfilled dependencies to occupy a single cell in the chart. A conjunctive node is formed when two single cells in the chart are merged into another cell, as occurs when two categories are being combined. Accordingly, the number of conjunctive nodes is an indication of the quantity of category combinations performed by the parser, and hence the degree of ambiguity encountered in parsing a given sentence as well as the physical size in memory of the chart data structure (Clark and Curran, 2007).

We also perform the standard evaluation 25 times on the development and test sections to compute the average wall-clock time for each experiment.

For the sake of comparison, we also perform the above experiments using our transformed corpus, but without barring the consideration of the now-redundant rules in the parser. In the results table, we refer to this ensemble as *New**.

7 Results

Figure 10 shows that the comma-normalised corpus, in concert with our modifications to the CCG parser removing support for the now-redundant rules, outperforms the baseline in ambiguity and parsing speed without adversely affecting parser performance or corpus coverage. This supports our claim in Section 4 that modifying only absorption commas does not remove any useful information from the corpus.

Why does comma normalisation have such a strong effect on parser efficiency? Any absorption rule engenders considerable ambiguity: consider that a comma can be attached at any internal level in the derivation. Although the C&C parser reins in this ambiguity by restricting the categories with which punctuation can combine, the effect of this attachment ambiguity is prominent when we consider the number of CCGbank sentences containing punctuation tokens apart from the full stop (32636 of 48934, or 66.7% of CCGbank derivations). Our changes to the parser reduce the number of comma absorption rules from 19 to 9, considerably reducing parser ambiguity in these sentences.

	Experiment	Labelled *P*	Labelled *R*	Labelled *F*	Auto *F*	Sent. acc.	Unlabelled *P*	Unlabelled *R*	Unlabelled *F*	Cat. acc. %	Covg %
Dev. set:	*New*	85.60	84.79	**85.19**	**83.39**	31.65	92.40	91.52	**91.96**	**93.08**	**99.11**
	Baseline	85.52	84.69	85.11	83.36	**31.93**	92.37	91.47	91.92	93.02	99.06
Test set:	*New*	86.17	85.48	**85.82**	**83.45**	**35.14**	92.33	91.59	**91.96**	**93.38**	**99.67**
	Baseline	86.03	85.37	85.70	83.24	34.70	92.28	91.57	91.93	93.27	99.63

(a) Standard CCGbank evaluation on development and test sets

	Experiment	Average parsing time	Parsing rates *sents/s*	Parsing rates *words/s*	Avg. conj. nodes	Avg. total nodes
Dev. set:	*New*	**89.49s**	**21.38**	**507.61**	**2472.77**	**5260.84**
	*New**	109.97s	17.40	413.05	3446.41	7800.58
	Baseline	112.73s	16.97	402.93	3349.94	7662.95
Test set:	*New*	**56.28s**	**42.78**	**984.05**	**2112.78**	**3653.83**
	*New**	88.30s	27.26	627.08	3087.54	6779.42
	Baseline	89.32s	26.95	619.97	3010.34	6590.26

(b) Parsing time on development and test sets

Figure 10: The effects of comma normalisation on parser accuracy and speed

An unexpected result is that the new corpus trained on the old parser (experiment *New**) results in *greater* ambiguity compared to the baseline. We determined that this is caused by our naïve choice of comma attachment level (as high as possible in the tree, the most general position). Our transformations have inadvertently added a small number of new rules to the corpus (with reference to Figure 8, this will occur when $(X\ ,\ \rightarrow X)$ is not attested in the corpus). On examination, in all of these newly added rules, the non-comma argument is the result of a *unary type-change* rule (such as $NP \rightarrow S/(S\backslash NP)$). We note by way of future work that we can eliminate these added rules in this way: if attaching as high as possible (our current criterion) would yield such a new rule, then attach the comma at its child instead (before such a rule has been applied).

We have developed a comma-normalised corpus which explains the same data with fewer rules, while slightly improving the coverage, accuracy and parsing time of a parser trained on this improved corpus. The resulting version of CCGbank treats the ubiquitous comma consistently, a desirable attribute for any NLP task which uses this valuable resource.

8 Conclusion

We believe that further improvements in parsing speed and memory consumption are possible by changing the representation of comma structures in the corpus. As we observed in Section 7, when the now-redundant rules are not disabled, parser ambiguity actually slightly *exceeds* the baseline. Changing the comma attachment level criterion may further improve parser ambiguity and memory consumption by reducing the number of rules the transformation adds to the treebank.

Far from being second-class, the correct analysis and treatment of punctuation in corpora and parsers has practical ramifications. We would like to continue to explore punctuation-awareness in parsing in the vein of Djordjevic et al. (2007), but from the viewpoint of corpus design. Can we bring the benefits of Nunberg's text grammar to a regular treebank by superimposing text grammar constituents onto those of the usual lexical grammar?

We would also like to explore the cross-linguistic treatment of punctuation in treebanks to inform possible improvements to existing corpora and derive guidelines for the design of future corpora.

We have eliminated a source of systematic inconsistency in a wide-coverage CCG treebank and simplified the implementation of a CCG parser, obtaining considerable improvements in speed and memory usage without sacrificing parser accuracy, demonstrating the importance of principled, consistent annotation in corpus design.

References

Murat Bayraktar, Bilge Say, and Varol Akman. 1998. An Analysis of English Punctuation: The Special Case of Comma. *International Journal of Corpus Linguistics*, 3(1):33–57.

Ann Bies, Mark Ferguson, Karen Katz, Robert MacIntyre, Victoria Tredinnick, Grace Kim, Mary Ann Marcinkiewicz, and Britt Schasberger. 1995. Bracketing Guidelines for Treebank II Style Penn Treebank Project. *University of Pennsylvania, Philadelphia.*

Ted Briscoe. 1994. Parsing (with) Punctuation etc. *Research Paper, Rank Xerox Research Centre, Grenoble.*

Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 104–111. Association for Computational Linguistics, Barcelona, Spain.

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

Bojan Djordjevic, James R. Curran, and Stephen Clark. 2007. Improving the Efficiency of a Wide-Coverage CCG Parser. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 39–47. Association for Computational Linguistics.

Julia Hockenmaier and Mark Steedman. 2005. CCGbank: Users manual. *University of Pennsylvania, Philadelphia.*

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

Bernard Jones. 1997. *What's the Point? A (Computational) Theory of Punctuations*. Ph.D. thesis, PhD thesis, Centre for Cognitive Science, University of Edinburgh, Edinburgh, UK.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Geoffrey Nunberg. 1990. *The Linguistics of Punctuation.* Center for the Study of Language and Information.

Miles Osborne. 1995. Can punctuation help learning? *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing, Lecture Notes in Artificial Intelligence*, pages 399–412.

Satoshi Sekine and Michael J. Collins. 2006. Evalb: a parseval crossing brackets evaluation script. URL `http://www.cs.nyu.edu/cs/projects/proteus/evalb`, accessed 10 June 2008.

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

Daniel Tse and James R. Curran. 2007. Extending CCGbank with quotes and multi-modal CCG. *Australasian Language Technology Workshop 2007*, pages 149–151.