Improving the Efficiency of a Wide-Coverage CCG Parser

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

Stephen Clark
Computing Laboratory
Oxford University
Wolfson Building, Parks Road
Oxford, OX1 3QD, UK
stephen.clark@comlab.ox.ac.uk

Abstract

The C&C CCG parser is a highly efficient linguistically motivated parser. The efficiency is achieved using a tightly-integrated supertagger, which assigns CCG lexical categories to words in a sentence. The integration allows the parser to request more categories if it cannot find a spanning analysis. We present several enhancements to the CKY chart parsing algorithm used by the parser. The first proposal is *chart repair*, which allows the chart to be efficiently updated by adding lexical categories individually, and we evaluate several strategies for adding these categories. The second proposal is to add *constraints* to the chart which require certain spans to be constituents. Finally, we propose *partial* beam search to further reduce the search space. Overall, the parsing speed is improved by over 35% with negligible loss of accuracy or coverage.

1 Introduction

A recent theme in parsing research has been the application of statistical methods to linguistically motivated grammars, for example LFG (Kaplan et al., 2004; Cahill et al., 2004), HPSG (Toutanova et al., 2002; Malouf and van Noord, 2004), TAG (Sarkar and Joshi, 2003) and CCG (Hockenmaier and Steedman, 2002; Clark and Curran, 2004b). The attraction of linguistically motivated parsers is the potential to produce rich output, in particular the predicate-argument structure representing the underlying meaning of a sentence. The disadvantage of such parsers is that they are typically not very efficient, parsing a few sentences per second on commodity hardware (Kaplan et al., 2004). The C&C CCG parser (Clark and Curran, 2004b) is an order of magnitude faster, but is still limited to around 25 sentences per second.

The key to efficient CCG parsing is a finite-state *supertagger* which performs much of the parsing work (Bangalore and Joshi, 1999). CCG is a lexicalised grammar formalism, in which elementary syntactic structures — in CCG's case *lexical categories* expressing subcategorisation information — are assigned to the words in a sentence. CCG supertagging can be performed accurately and efficiently by a Maximum Entropy tagger (Clark and Curran, 2004a). Since the lexical categories contain so much grammatical information, assigning them with low average ambiguity leaves the parser, which combines them together, with much less work to do at parse time. Hence Bangalore and Joshi (1999), in the context of LTAG parsing, refer to supertagging as *almost parsing*.

Clark and Curran (2004a) presents a novel method of integrating the supertagger and parser: initially only a small number of categories, on average, is assigned to each word, and the parser attempts to find a spanning analysis using the CKY chart-parsing algorithm. If one cannot be found, the parser requests more categories from the supertagger and builds the chart again from scratch. This process repeats until the parser is able to build a chart containing a spanning analysis.[1]

[1] Tsuruoka and Tsujii (2004) investigate a similar idea in the context of the CKY algorithm for a PCFG.

Proceedings of the 10th Conference on Parsing Technologies, pages 39–47,

The supertagging accuracy is high enough that the parser fails to find a spanning analysis using the initial category assignment in approximately 4% of Wall Street Journal sentences (**?**). However, parsing this 4%, which largely consists of the longer sentences, is disproportionately expensive.

This paper describes several modifications to the C&C parser which improve parsing efficiency without reducing accuracy or coverage by reducing the impact of the longer sentences. The first involves *chart repair*, where the CKY chart is repaired when extra lexical categories are added (according to the scheme described above), instead of being rebuilt from scratch. This allows an even tighter integration of the supertagger, in that the parser is able to request *individual* categories. We explore methods for choosing which individual categories to add, resulting in an 11% speed improvement.

The next modification involves parsing with *constraints*, so that certain spans are required to be constituents. This reduces the search space considerably by eliminating a large number of constituents which cross the boundaries of these spans. The best set of constraints results in a 10% speed improvement over the original parser. These constraints are general enough that they could be applied to any constituency-based parser. Finally, we experiment with several beam strategies to reduce the search space, finding that a *partial beam* which operates on part of the chart is most effective, giving a further 6.1% efficiency improvement.

The chart repair and constraints interact in an interesting, and unexpected, manner when combined, giving a 35.7% speed improvement overall without any loss in accuracy or coverage. This speed improvement is particularly impressive because it involves techniques which only apply to 4% of Wall Street Journal sentences.

2 The CCG Parser

Clark and Curran (2004b) describes the CCG parser. The grammar used by the parser is extracted from CCGbank, a CCG version of the Penn Treebank (Hockenmaier, 2003). The grammar consists of 425 lexical categories plus a small number of combinatory rules which combine the categories (Steedman, 2000). A Maximum Entropy supertagger first assigns lexical categories to the words in a sentence, which are then combined by the parser using the combinatory rules. A log-linear model scores the alternative parses. We use the normal-form model, which assigns probabilities to single derivations based on the normal-form derivations in CCGbank. The features in the model are defined over local parts of the derivation and include word-word dependencies. A packed chart representation allows efficient decoding, with the Viterbi algorithm finding the most probable derivation.

The supertagger uses a log-linear model to define a distribution over the lexical category set for each word and the previous two categories (Ratnaparkhi, 1996) and the forward backward algorithm efficiently sums over all histories to give a distribution for each word. These distributions are then used to assign a set of lexical categories to each word (**?**). The number of categories in each set is determined by a parameter β: all categories are assigned whose forward-backward probabilities are within β of the highest probability category (**?**). If the parser cannot then find a spanning analysis, the value of β is reduced — so that more lexical categories are assigned — and the parser tries again. This process repeats until an analysis spanning the whole sentence is found.

In our previous work, when the parser was unable to find a spanning analysis, the chart was destroyed and then rebuilt from scratch with more lexical categories assigned to each word. However, this rebuilding process is wasteful because the new chart is always a superset of the old one and could be created by just updating the previous chart. We describe the *chart repair* process in Section 3 which allows additional categories to be assigned to an existing chart and the CKY algorithm run over just those parts of the chart which require modification.

2.1 Chart Parsing

The parser uses the CKY chart parsing algorithm (Kasami, 1965; Younger, 1967) described in Steedman (2000). The CKY algorithm applies naturally to CCG since the grammar is binary. It builds the chart bottom-up, starting with the lexical categories spanning single words, incrementally increasing the span until the whole sentence is covered. Since the constituents are built in order of span size, at every stage all the sub-constituents which could be used to create a particular new constituent are already present in the chart.

The charts are *packed* by grouping together equivalent chart entries, which allows a large number of derivations to be represented efficiently. Entries are *equivalent* when they interact in the same manner with both the generation of subsequent parse structure and the statistical parse selection. In practice, this means that equivalent entries have the same span; form the same structures, i.e. the remaining derivation plus dependencies, in any subsequent parsing; and generate the same features in any subsequent parsing.

The Viterbi algorithm is used to find the most probable derivation from a packed chart. For each equivalence class of individual entries, we record the entry at the root of the subderivation which has the highest score for the class. The equivalence classes are defined so that any other individual entry cannot be part of the highest scoring derivation for the sentence. The highest-scoring subderivations can be calculated recursively using the highest-scoring equivalence classes that were combined to create the individual entry.

Given a sentence of n words, we define $pos \in \{0, \ldots, n-1\}$ to be the starting position of an entry in the chart (represented by a CCG category) and $span \in \{1, \ldots, n\}$ its length. Let $\text{cell}(pos, span)$ be the set of categories which span the sentence from pos to $pos + span$. These will be combinations of categories in $\text{cell}(pos, k)$ and $\text{cell}(pos+k, span-k)$ for all $k \in \{1, \ldots, span-1\}$. The chart is a two dimensional array indexed by pos and $span$. The valid $(pos, span)$ pairs correspond to $pos + span \leq n$, that is, to spans that do not extend beyond the end of the sentence. The squares represent valid cells in Figure 1. The span from position 3 with length 4, i.e. $\text{cell}(3, 4)$, is marked with a diamond in Figure 2.

3 Chart Repair

The parser interacts with the supertagger by decreasing the value of the β parameter when a spanning analysis cannot be found for a sentence. This has the effect of adding more lexical categories to the chart. Instead of rebuilding the chart from scratch when new categories are added, it can be repaired

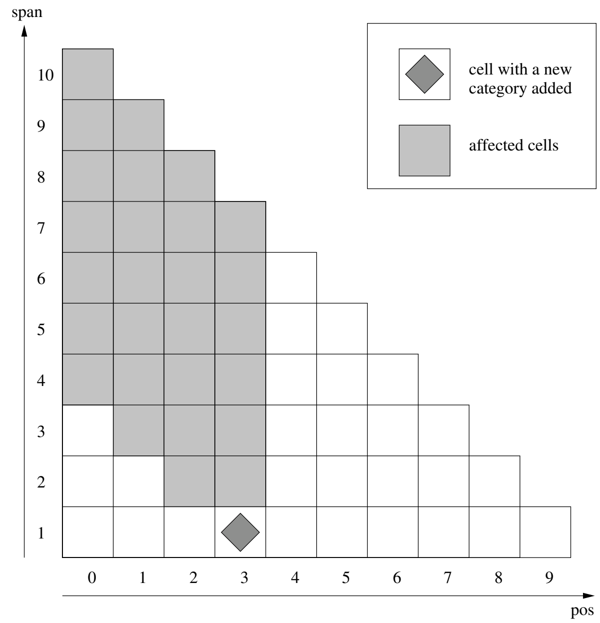

Figure 1: Cells affected by chart repair.

by modifying cells that are affected by the new categories. Considering the case where a single lexical category is added to the ith word in an n word sentence, the new category can only affect the cells that satisfy $pos \leq i$ and $pos + span > i$. These cells are shown in Figure 1 for the word at position 3.

The number of affected cells is $(n-pos)(pos+1)$, and so the average over the sentence is approximately $\frac{1}{n}\int_0^{n-1}(n-p)(p+1)\ dp \approx \frac{n^2}{6}$ cells. The total number of cells in the chart is $\frac{n(n+1)}{2}$. The chart can therefore be repaired bottom up, in CKY order, by updating a third of the cells on average.

Additional lexical categories for a word are inserted into the corresponding cell in the bottom row, with the additional categories being marked as new. For each cell C in the second row, each pair of cells A and B is considered whose spans combine to create the span of C. In the original CKY, all categories from A are combined with all categories from B. In chart repair, categories are only combined if at least one of them is new, because otherwise the result is already in C. The categories added to C are marked, and the process is repeated for all affected cells in CKY order.

Chart repair speeds up parsing for two reasons. First, it reuses previous computations and eliminates wasteful rebuilding of the chart. Second, it allows lexical categories to be added to the chart *one at a*

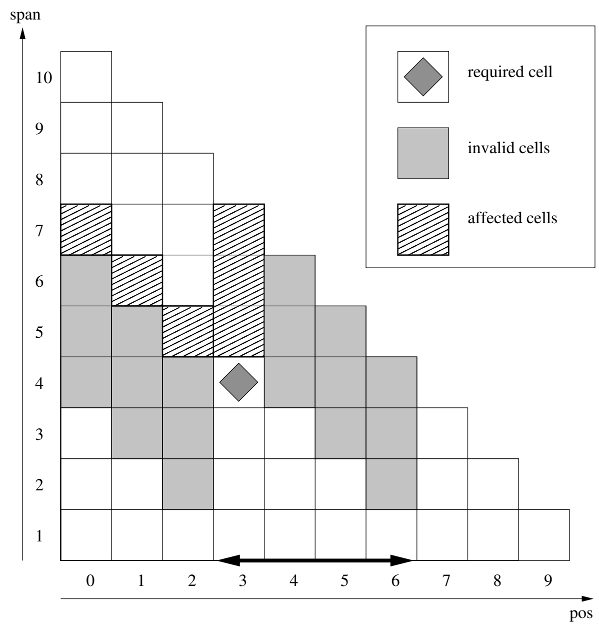

Figure 2: Cells affected by adding a constraint.

time until a spanning derivation is found. In the original approach extra categories were added in bulk by changing the β level, which significantly increased the average ambiguity. Chart repair allows the minimum amount of ambiguity to be added for a spanning derivation to be found.

The C&C parser has a predefined limit on the number of categories in the chart. If this is exceeded before a spanning analysis is found then the parser fails on the sentence. Our new strategy allows a chart containing a spanning analysis to be built with the minimum number of categories possible. This means that some sentences can now be parsed that would have previously exceeded the limit, slightly increasing coverage.

3.1 Category selection

The order in which lexical categories are added to the chart will impact on parsing speed and accuracy, and so we evaluate several alternatives. The first ordering (β VALUE) is by decreasing β value, where the β value is the ratio between the probability of the most likely category and the probability of the given category for that word.[2] The second ordering (PROB) is by decreasing category probability as assigned by the supertagger using the forward-backward algorithm.

We also investigated ordering categories using information from the chart. Examining the sentences which required chart repair showed that, when a word is missing the correct category, the cells affected (as defined in Section 3) by the cell are often empty. The CHART ordering uses this observation to select the next lexical category to assign. It selects the word corresponding to the cell with the highest number of empty affected cells, and then adds the highest probability category not in the chart for that word. Finally, we included a RANDOM ordering baseline for comparison purposes.

4 Constraints

The set of possible derivations can be constrained if we know in advance that a particular span is *required* to be the yield of a single constituent in the correct parse. A *constraint* on span p reduces the search space because p must be the yield of a single cell. This means that cells with yields that cross the boundary of p cannot be part of a correct derivation, and do not need to be considered (the grey cells in Figure 2). In addition, if a cell yields p as a *prefix* or *suffix* (the hashed cells in Figure 2) then it also has constraints on how it can be created.

Figure 2 shows an example constraint requiring words 3–6 to be a constituent, which corresponds to $p = \text{cell}(3,4)$. Consider $\text{cell}(3,7)$: it yields words 3–9 and so contains p as the prefix. Normally it can be created by combining $\text{cell}(3,1)$ with $\text{cell}(4,6)$, or $\text{cell}(3,2)$ with $\text{cell}(5,5)$, and so on up to $\text{cell}(3,6)$ with $\text{cell}(9,1)$. However the first three combinations are not allowed because the second child crosses the boundary of p. This gives a lower limit for the span of the left child. Similarly, if p is the suffix of the span of a cell then there is a lower limit on the span of the right child.

As the example demonstrates, a single constraint can eliminate many combinations, reducing the search space significantly, and thus improving parsing efficiency.

4.1 Creating Constraints

How can we know in advance that the correct derivation must yield specific spans, since this appears to require knowledge of the parse itself? We have ex-

[2] We are overloading the use of β for convenience. Here, β refers to the variable ratio dependent on the particular category, whereas the β value used in supertagging is a *cutoff* applied to the variable ratio.

plored constraints derived from shallow parsing and from the raw sentence. Our results demonstrate that simple constraints can reduce parsing time significantly without loss of coverage or accuracy.

Chunk tags were used to create constraints. We experimented with both gold standard chunks from the Penn Treebank and also chunker output from the C&C chunk tagger. The tagger is very similar to the Maximum Entropy POS tagger described in Curran and Clark (2003). Only NP chunks were used because the accuracy of the tagger for other chunks is lower. The Penn Treebank chunks required modification because CCGbank analyses some constructions differently. We also created longer NPs by concatenating adjacent base NPs, for example in the case of possessives.

A number of *punctuation constraints* were used and had a significant impact especially for longer sentences. There are a number of punctuation rules in CCGbank which absorb a punctuation mark by combining it with a category and returning a category of the same type. These rules are very productive, combining with many constituent types. However, in CCGbank the sentence final punctuation is always attached at the root. A constraint on the first $n - 1$ words was added to force the parser to only attach the sentence final punctuation once the rest of the sentence has been parsed.

Constraints are placed around parenthesised and quoted phrases that usually form constituents before attaching elsewhere. Constraints are also placed around phrases bound by colons, semicolons, or hyphens. These constraints are especially effective for long sentences with many clauses separated by semicolons, reducing the sentence to a number of smaller units which significantly improves parsing efficiency.

In some instances, adding constraints can be harmful to parsing efficiency and/or accuracy. Lack of precision in the constraints can come from noisy output from NLP components, e.g. the chunker, or from rules which are not always applicable, e.g. punctuation constraints. We find that the punctuation constraints are particularly effective while the gold standard chunks are required to gain any benefit for the NP constraints. Adding constraints also has the potential to increase coverage because the reduced search space means that longer sentences can be parsed without exceeding the pre-defined limits on chart size.

5 Selective Beam Search

Beam search involves greedy elimination of low probability partial derivations before they can form complete derivations. It is used in many parsers to reduce the search space, for example Collins (2003). We use a *variable width beam* where all categories c in a particular cell C that satisfy $\mathrm{score}(c) < \max\{\mathrm{score}(x) | x \in C\} - B$, for some beam cutoff B, are removed. The category scores $\mathrm{score}(c)$ are log probabilities.

In the C&C parser, the entire packed chart is constructed first and then the spanning derivations are marked. Only the partial derivations that form part of spanning derivations are scored to select the best parse, which is a small fraction of the categories in the chart. Because the categories are scored with a complex statistical model with a large number of features, the time spent calculating scores is significant. We found that applying a beam to every cell during the construction of the chart was more expensive than not using the beam at all. When the beam was made harsh enough to be worthwhile, it reduced accuracy and coverage significantly.

We propose *selective beam search* where the beam is only applied to spans of particular lengths. The shorter spans are most important to cull because there are many more of them and removing them has the largest impact in terms of reducing the search space. However, the supertagger already acts like a beam at the lexical category level and the parser model has fewer features at this level, so the beam may be more accurate for longer spans. We therefore expect the beam to be most effective for spans of intermediate length.

6 Experiments

The parser was trained on CCGbank sections 02-21 and section 00 was used for development. The performance is measured in terms of coverage, F-score and parsing time. The F-score is for labelled dependencies compared against the predicate-argument dependencies in CCGbank. The time reported includes loading the grammar and statistical model, which takes around 5 seconds, and parsing the 1913 sentences in section 00.

The failure rate (opposite of coverage) is broken down into sentences with length ≤ 40 and > 40 because longer sentences are more difficult to parse and the C&C parser already has very high coverage on shorter sentences. There are 1784 1-40 word sentences and 129 41+ word sentences. The average length and standard deviation in the 41+ set are 50.8 and 31.5 respectively.

All experiments used gold standard POS tags. Original and REPAIR do not use constraints. The NP(GOLD) experiments use Penn Treebank gold standard NP chunks to determine an upper bound on the utility of chunk constraints. The times reported for NP(C&C) using the C&C chunker include the time to load the chunker model and run the chunker (around 1.3 seconds). PUNCT adds all of the punctuation constraints.

Finally the best system was compared against the original parser on section 23, which has 2257 sentences of length 1-40 and 153 of length 41+. The maximum length is only 65, which explains the high coverage for the 41+ section.

6.1 Chart Repair Results

The results in Table 1 show that chart repair gives an immediate 11.1% improvement in speed and a small 0.21% improvement in accuracy. 96.1% of sentences do not require chart repair because they are successfully parsed using the initial set of lexical categories supplied by the supertagger. Hence, 11% is a significant improvement for less than 4% of the sentences.

We believe the accuracy was improved (on top of the efficiency) because of the way the repair process adds new categories. Adding categories individually allows the parser to be influenced by the probabilities which the supertagger assigns, which are not directly modelled in the parser. If we were to add this information from the supertagger into the parser statistical model directly we would expect almost no accuracy difference between the original method and chart repair.

Table 2 shows the impact of different category ordering approaches for chart repair (with PUNCT constraints). The most effective approach is to use the information from the chart about the proportion of empty cells, which adds half as many categories on average as the β value and probability based approaches. All of our approaches significantly outperform randomly selecting extra categories. The CHART category ordering is used for the remaining experiments.

METHOD	secs	%	F-SCORE	CATS
RANDOM	70.2	-16.2	86.57	23.1
β VALUE	60.4	—	**86.66**	15.7
PROB	60.1	0.5	86.65	14.3
CHART	**57.2**	**5.3**	86.61	**7.0**

Table 2: Category ordering for chart repair.

6.2 Constraints Results

The results in Table 1 show that, without chart repair, using gold standard noun phrases does not improve efficiency, while using noun phrases identified by the C&C chunker decreases speed by 10.8%. They both also slightly reduce parsing accuracy. The number of times the parsing process had to be restarted with the constraints removed, was more costly than the reduction of the search space. This is unsurprising because the chunk data was not obtained from CCGbank and the chunker is not accurate enough for the constraints to improve parsing efficiency. The most frequent inconsistencies between CCGbank and chunks extracted from the Penn Treebank were fixed in a preprocessing step as explained in Section 4.1, but the less frequent constructions are still problematic.

The best results for parsing with constraints (without repair) were with *both* punctuation and gold standard noun phrase constraints, with 20.5% improvement in speed and 0.42% in coverage, but an F-score penalty of 0.3%. This demonstrates the possible efficiency gain with a perfect chunker – the corresponding results with the C&C chunker are still worse than without constraints. The best results without a decrease in accuracy use only punctuation constraints, with 10.4% increase in speed and 0.37% in coverage. The punctuation constraints also have the advantage of being simple to implement.

The best overall efficiency gain was obtained when punctuation and gold standard noun phrases were used with chart repair, with a 45.4% improvement in speed and 0.63% in coverage, and a 0.4% drop in accuracy. The best results without a drop in

METHOD	secs	%	F-SCORE	COVER	$n \leq 40$	$n > 40$
Original	88.3	—	86.54	98.85	0.392	11.63
REPAIR	78.5	11.1	**86.75**	99.01	0.336	10.08
NP(GOLD)	88.4	-0.1	86.27	99.06	0.224	10.85
NP(C&C)	97.8	-10.8	86.31	99.16	0.224	9.30
PUNCT	79.1	10.4	86.56	99.22	**0.168**	9.30
NP(GOLD) + PUNCT	69.8	20.5	86.24	99.27	**0.168**	8.53
NP(C&C) + PUNCT	97.0	-9.9	86.31	99.16	**0.168**	10.08
NP(GOLD) + REPAIR	65.0	26.4	86.04	99.37	0.224	6.20
NP(C&C) + REPAIR	77.5	12.2	86.35	99.37	0.224	6.20
PUNCT + REPAIR	57.2	35.2	86.61	99.48	**0.168**	5.43
NP(GOLD) + PUNCT + REPAIR	**48.2**	**45.4**	86.14	99.48	**0.168**	5.43
NP(C&C) + PUNCT + REPAIR	63.2	28.4	86.43	**99.53**	0.163	**3.88**

Table 1: Parsing performance on section 00 with constraints and chart repair

METHOD	secs	%	F-SCORE	COVER	$n \leq 40$	$n > 40$
Original	88.3	—	86.54	98.85	0.392	11.63
PUNCT	79.1	10.4	86.56	99.22	**0.168**	9.30
REPAIR	78.5	11.1	**86.75**	99.01	0.336	10.08
PUNCT + REPAIR	57.2	35.2	86.61	**99.48**	**0.168**	**5.43**
PUNCT + REPAIR + BEAM	**52.4**	**40.7**	86.56	**99.48**	**0.168**	**5.43**

Table 3: Best performance on Section 00

accuracy were with only punctuation constraints and chart repair, with improvements of 35.2% speed and 0.63% coverage. Coverage on both short and long sentences is improved – the best results show a 43% and 67% decrease in failure rate for sentence lengths in the ranges 1-40 and 41+ respectively.

6.3 Partial Beam Results

We found that using the selective beam on 1–2 word spans had negligible impact on speed and accuracy. Using the beam on 3–4 word spans had the most impact without accuracy penalty, improving efficiency by another $\sim$5%. Experiments with the selective beam on longer spans continued to improve efficiency, but with a much greater penalty in F-score, e.g. a further $\sim$5% at a cost of 0.5% F-score for 3–6 word spans. However, we are interested in efficiency improvements with negligible cost to accuracy.

6.4 Overall Results

Table 3 summarises the results for section 00. The chart repair and punctuation constraints individually increase parsing efficiency by around 10%. However, the most interesting result is that in combination they increase efficiency by over 35%. This is because the cost of rebuilding the chart when the constraints are incorrect has been significantly reduced by chart repair. Finally, the use of the selective beam gives modest improvement of 5.5%. The overall efficiency gain on section 00 is 40.7% with an additional 0.5% coverage, halving both the number of short and long sentences that fail to be parsed.

Table 4 shows the performance of the punctuation constraints, chart repair and selective beam system on section 23. The results are consistent with section 00, showing a 30.9% improvement in speed and 0.29% in coverage, with accuracy staying at roughly the same level. The results show a consistent 35-40% reduction in parsing time and a 40-65% reduction in parse failure rate.

7 Conclusion

We have introduced several modifications to CKY parsing for CCG that significantly increase parsing efficiency without an accuracy or coverage penalty.

METHOD	secs	%	F-SCORE	COVER	$n \leq 40$	$n > 40$
Original	91.3	—	**86.92**	99.29	0.621	1.961
PUNCT + REPAIR + BEAM	**58.7**	**35.7**	86.82	**99.58**	**0.399**	**0.654**

Table 4: Best performance on Section 23

Chart repair improves efficiency by reusing the chart from the previous parse attempts. This allows us to further tighten the parser-supertagger integration by adding one lexical category at a time until a spanning derivation is found. We have also explored several approaches to selecting which category to add next. We intend to further explore strategies for determining which category to add next when a parse fails. This includes combining chart and probability based orderings. Chart repair alone gives an 11.1% efficiency improvement.

Constraints improve efficiency by avoiding the construction of sub-derivations that will not be used. They have a significant impact on parsing speed and coverage without reducing the accuracy, provided the constraints are identified with sufficient precision. Punctuation constraints give a 10.4% improvement, but NP constraints require higher precision NP chunking than is currently available for CCGbank.

Constraints and chart repair both manipulate the chart for more efficient parsing. Adding categories one at a time using chart repair is almost a form of agenda-based parsing. We intend to explore other methods for pruning the space and agenda-based parsing, in particular A* parsing (Klein and Manning, 2003), which will allow only the most probable parts of the chart to be built, improving efficiency while still ensuring the optimal derivation is found.

When all of our modifications are used parsing speed increases by 35-40% and the failure rate decreases by 40-65%, both for sentences of length 1-40 and 41+, with a negligible accuracy penalty. The result is an even faster state-of-the-art wide-coverage CCG parser.

Acknowledgements

We would like to thank the anonymous reviewers for their feedback. James Curran was funded under ARC Discovery grants DP0453131 and DP0665973.

References

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

A. Cahill, M. Burke, R. O'Donovan, J. van Genabith, and A. Way. 2004. Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of the 42nd Meeting of the ACL*, pages 320–327, Barcelona, Spain.

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

Stephen Clark and James R. Curran. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proceedings of ACL-04*, pages 104–111, Barcelona, Spain.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

James R. Curran 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.

James R. Curran, Stephen Clark, and David Vadas. 2006. Multi-tagging for lexicalized-grammar parsing. In *Proceedings of COLING/ACL-06*, pages 697–704, Sydney, Austrailia.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*, pages 335–342, Philadelphia, PA.

Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Ron Kaplan, 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 the Human Language Technology Conference and the 4th Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'04)*, Boston, MA.

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

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of Human Language Technology and the North American Chapter of the Association for Computational Linguistics Conference*, pages 119–126, Edmond, Canada.

Robert Malouf and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop: Beyond shallow analyses - Formalisms and statistical modeling for deep analyses*, Hainan Island, China.

Adwait Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the EMNLP Conference*, pages 133–142, Philadelphia, PA.

Anoop Sarkar and Aravind Joshi. 2003. Tree-adjoining grammars and its application to statistical parsing. In Rens Bod, Remko Scha, and Khalil Sima'an, editors, *Data-oriented parsing*. CSLI.

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

Kristina Toutanova, Christopher Manning, Stuart Shieber, Dan Flickinger, and Stephan Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, pages 253–263, Sozopol, Bulgaria.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2004. Iterative cky parsing for probabilistic context-free grammars. In *Proceedings of the IJCNLP conference*, pages 52–60, Hainan Island, China.

D. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.