Dependency Hashing for n-best CCG Parsing

Dominick Ng and **James R. Curran**
ə-lab, School of Information Technologies
University of Sydney
NSW, 2006, Australia
{dominick.ng,james.r.curran}@sydney.edu.au

Abstract

Optimising for one grammatical representation, but evaluating over a different one is a particular challenge for parsers and n-best CCG parsing. We find that this mismatch causes many n-best CCG parses to be semantically equivalent, and describe a hashing technique that eliminates this problem, improving oracle n-best F-score by 0.7% and reranking accuracy by 0.4%. We also present a comprehensive analysis of errors made by the C&C CCG parser, providing the first breakdown of the impact of implementation decisions, such as supertagging, on parsing accuracy.

1 Introduction

Reranking techniques are commonly used for improving the accuracy of parsing (Charniak and Johnson, 2005). Efficient decoding of a parse forest is infeasible without dynamic programming, but this restricts features to local tree contexts. Reranking operates over a list of n-best parses according to the original model, allowing poor local parse decisions to be identified using arbitrary rich parse features.

The performance of reranking depends on the quality of the underlying n-best parses. Huang and Chiang (2005)'s n-best algorithms are used in a wide variety of parsers, including an n-best version of the C&C CCG parser (Clark and Curran, 2007; Brennan, 2008). The oracle F-score of this parser (calculated by selecting the most optimal parse in the n-best list) is 92.60% with $n = 50$ over a baseline 1-best F-score of 86.84%. In contrast, the Charniak parser records an oracle F-score of 96.80% in 50-best mode over a baseline of 91.00% (Charniak and Johnson, 2005). The 4.2% oracle score difference suggests that further optimisations may be possible for CCG.

We describe how n-best parsing algorithms that operate over derivations do not account for absorption ambiguities in parsing, causing semantically identical parses to exist in the CCG n-best list. This is caused by the mismatch between the optimisation target (different derivations) and the evaluation target (CCG dependencies). We develop a hashing technique over dependencies that removes duplicates and improves the oracle F-score by 0.7% to 93.32% and reranking accuracy by 0.4%. Huang et al. (2006) proposed a similar idea where strings generated by a syntax-based MT rescoring system were hashed to prevent duplicate translations.

Despite this improvement, there is still a substantial gap between the C&C and Charniak oracle F-scores. We perform a comprehensive subtractive analysis of the C&C parsing pipeline, identifying the relative contribution of each error class and why the gap exists. The parser scores 99.49% F-score with gold-standard categories on section 00 of CCGbank, and 94.32% F-score when returning the best parse in the chart using the supertagger on standard settings. Thus the supertagger contributes roughly 5% of parser error, and the parser model the remaining 7.5%. Various other speed optimisations also detrimentally affect accuracy to a smaller degree.

Several subtle trade-offs are made in parsers between speed and accuracy, but their actual impact is often unclear. Our work investigates these and the general issue of how different optimisation and evaluation targets can affect parsing performance.

Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 497–505,
Jeju, Republic of Korea, 8-14 July 2012.

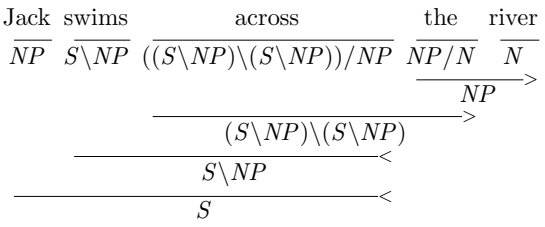

Figure 1: A CCG derivation with a PP adjunct, demonstrating forward and backward combinator application. Adapted from Villavicencio (2002).

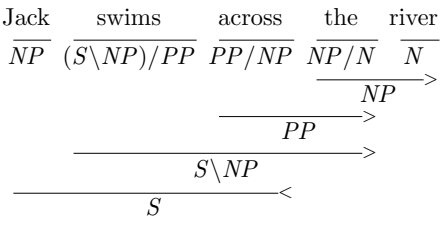

Figure 2: A CCG derivation with a PP argument (note the categories of *swims* and *across*). The bracketing is identical to Figure 1, but nearly all dependencies have changed.

2 Background

Combinatory Categorial Grammar (CCG, Steedman, 2000) is a lexicalised grammar formalism based on formal logic. The grammar is directly encoded in the lexicon in the form of *categories* that govern the syntactic behaviour of each word.

Atomic categories such as N (noun), NP (noun phrase), and PP (prepositional phrase) represent complete units. Complex categories encode subcategorisation information and are functors of the form X/Y or $X\backslash Y$. They represent structures which combine with an argument category Y to produce a result category X. In Figure 1, the complex category $S\backslash NP$ for *swims* represents an intransitive verb requiring a subject NP to the left.

Combinatory rules are used to combine categories together to form an analysis. The simplest rules are forward and backward application, where complex categories combine with their outermost arguments. Forward and backward composition allow categories to be combined in a non-canonical order, and type-raising turns a category into a higher-order functor. A ternary coordination rule combines two identical categories separated by a $conj$ into one.

As complex categories are combined with their arguments, they create a logical form representing the syntactic and semantic properties of the sentence. This logical form can be expressed in many ways; we will focus on the *dependency* representation used in CCGbank (Hockenmaier and Steedman, 2007). In Figure 1, *swims* generates one dependency:

$$\langle \textit{swims},\ S[dcl]\backslash NP_1,\ 1,\ \textit{Jack},\ - \rangle$$

where the dependency contains the head word, head category, argument slot, argument word, and whether the dependency is long-range.

2.1 Corpora and evaluation

CCGbank (Hockenmaier, 2003) is a transformation of the Penn Treebank (PTB) data into CCG derivations, and it is the standard corpus for English CCG parsing. Other CCG corpora have been induced in a similar way for German (Hockenmaier, 2006) and Chinese (Tse and Curran, 2010). CCGbank contains 99.44% of the sentences from the PTB, and several non-standard rules were necessary to achieve this coverage. These include punctuation absorption rules and unary type-changing rules for clausal adjuncts that are otherwise difficult to represent.

The standard CCG parsing evaluation calculates labeled precision, recall, and F-score over the dependencies recovered by a parser as compared to CCGbank (Clark et al., 2002). All components of a dependency must match the gold standard for it to be scored as correct, and this makes the procedure much harsher than the PARSEVAL labeled brackets metric. In Figure 2, the PP *across the river* has been interpreted as an argument rather than an adjunct as in Figure 1. Both parses would score identically under PARSEVAL as their bracketing is unchanged. However, the adjunct to argument change results in different categories for *swims* and *across*; nearly every CCG dependency in the sentence is headed by one of these two words and thus each one changes as a result. An incorrect argument/adjunct distinction in this sentence produces a score close to 0.

All experiments in this paper use the normal-form C&C parser model over CCGbank 00 (Clark and Curran, 2007). Scores are reported for sentences which the parser could analyse; we observed similar conclusions when repeating our experiments over the subset of sentences that were parsable under all configurations described in this paper.

2.2 The C&C parser

The C&C parser (Clark and Curran, 2007) is a fast and accurate CCG parser trained on CCGbank 02-21, with an accuracy of 86.84% on CCGbank 00 with the normal-form model. It is a two-phase system, where a supertagger assigns possible categories to words in a sentence and the parser combines them using the CKY algorithm. An n-best version incorporating the Huang and Chiang (2005) algorithms has been developed (Brennan, 2008). Recent work on a softmax-margin loss function and integrated supertagging via belief propagation has improved this to 88.58% (Auli and Lopez, 2011).

A parameter β is passed to the supertagger as a multi-tagging probability beam. β is initially set at a very restrictive value, and if the parser cannot form an analysis the supertagger is rerun with a lower β, returning more categories and giving the parser more options in constructing a parse. This *adaptive* supertagging prunes the search space whilst maintaining coverage of over 99%.

The supertagger also uses a *tag dictionary*, as described by Ratnaparkhi (1996), and accepts a cutoff k. Words seen more than k times in CCGbank 02-21 may only be assigned categories seen with that word more than 5 times in CCGbank 02-21; the frequency must also be no less than 1/500th of the most frequent tag for that word. Words seen fewer than k times may only be assigned categories seen with the POS of the word in CCGbank 02-21, subject to the cutoff and ratio constraint (Clark and Curran, 2004b). The tag dictionary eliminates infrequent categories and improves the performance of the supertagger, but at the cost of removing unseen or infrequently seen categories from consideration.

The parser accepts POS-tagged text as input; unlike many PTB parsers, these tags are fixed and remain unchanged throughout during the parsing pipeline. The POS tags are important features for the supertagger; parsing accuracy using gold-standard POS tags is typically 2% higher than using automatically assigned POS tags (Clark and Curran, 2004b).

2.3 n-best parsing and reranking

Most parsers use dynamic programming, discarding infeasible states in order to maintain tractability. However, constructing an n-best list requires keeping the top n states throughout. Huang and Chiang (2005) define several n-best algorithms that allow dynamic programming to be retained whilst generating precisely the top n parses – using the observation that once the 1-best parse is generated, the 2nd best parse must differ in exactly one location from it, and so forth. These algorithms are defined on a hypergraph framework equivalent to a chart, so the parses are distinguished based on their derivations. Huang et al. (2006) develop a translation reranking model using these n-best algorithms, but faced the issue of different derivations yielding the same string. This was overcome by storing a hashtable of strings at each node in the tree, and rejecting any derivations that yielded a previously seen string.

Collins (2000)'s parser reranker uses n-best parses of PTB 02-21 as training data. Reranker features include lexical heads and the distances between them, context-free rules in the tree, n-grams and their ancestors, and parent-grandparent relationships. The system improves the accuracy of the Collins parser from 88.20% to 89.75%.

Charniak and Johnson (2005)'s reranker uses a similar setup to the Collins reranker, but utilises much higher quality n-best parses. Additional features on top of those from the Collins reranker such as subject-verb agreement, n-gram local trees, and right-branching factors are also used. In 50-best mode the parser has an oracle F-score of 96.80%, and the reranker produces a final F-score of 91.00% (compared to an 89.70% baseline).

3 Ambiguity in n-best CCG parsing

The type-raising and composition combinators allow the same logical form to be created from different category combination orders in a derivation. This is termed *spurious ambiguity*, where different derivational structures are semantically equivalent and will evaluate identically despite having a different phrase structure. The C&C parser employs the normal-form constraints of Eisner (1996) to address spurious ambiguity in 1-best parsing.

Absorption ambiguity occurs when a constituent may be legally placed at more than one location in a derivation, and all of the resulting derivations are semantically equivalent. Punctuation such as commas, brackets, and periods are particularly prone to

	Avg P/sent	Distinct P/sent	% Distinct
10-best	9.8	5.1	52
50-best	47.6	16.0	34
10-best$^{\#}$	9.0	9.0	100
50-best$^{\#}$	37.9	37.9	100

Table 1: Average and distinct parses per sentence over CCGbank 00 with respect to CCG dependencies. # indicates the inclusion of dependency hashing

	Avg P/sent	Distinct P/sent	% Distinct
10-best	9.8	4.4	45
50-best	47.6	13.0	27
10-best$^{\#}$	8.9	8.1	91
50-best$^{\#}$	37.1	31.5	85

Table 2: Average and distinct parses per sentence over CCGbank 00 with respect to GRs. # indicates the inclusion of dependency hashing

absorption ambiguity in CCG; Figure 3 depicts four semantically equivalent sequences of absorption and combinator application in a sentence fragment.

The Brennan (2008) CCG n-best parser differentiates CCG parses by derivation rather than logical form. To illustrate how this is insufficient, we ran the parser using Algorithm 3 of Huang and Chiang (2005) with $n = 10$ and $n = 50$, and calculated how many parses were semantically distinct (i.e. yield different dependencies). The results (summarised in Table 1) are striking: just 52% of 10-best parses and 34% of 50-best parses are distinct. We can also see that fewer than n parses are found on average for each sentence; this is mostly due to shorter sentences that may only receive one or two parses.

We perform the same diversity experiment using the DepBank-style grammatical relations (GRs, King et al., 2003; Briscoe and Carroll, 2006) output of the parser. GRs are generated via a dependency to GR mapping in the parser as well as a post-processing script to clean up common errors (Clark and Curran, 2007). GRs provide a more formalism-neutral comparison and abstract away from the raw CCG dependencies; for example, in Figures 1 and 2, the dependency from *swims* to *Jack* would be abstracted into `(subj swims Jack)` and thus would be identical in both parses. Hence, there are even fewer distinct parses in the GR results summarised in Table 2: 45% and 27% of 10-best and 50-best parses respectively yield unique GRs.

3.1 Dependency hashing

To address this problem of semantically equivalent n-best parses, we define a uniqueness constraint over all the n-best candidates:

Constraint. *At any point in the derivation, any n-best candidate must not have the same dependencies as any candidate already in the list.*

Enforcing this constraint is non-trivial as it is infeasible to directly compare every dependency in a partial tree with another. Due to the flexible notion of constituency in CCG, dependencies can be generated at a variety of locations in a derivation and in a variety of orders. This means that comparing all of the dependencies in a particular state may require traversing the entire sub-derivation at that point. Parsing is already a computationally expensive process, so we require as little overhead from this check as possible.

Instead, we represent all of the CCG dependencies in a sub-derivation using a hash value. This allows us to compare the dependencies in two derivations with a single numeric equality check rather than a full iteration. The underlying idea is similar to that of Huang et al. (2006), who maintain a hashtable of unique strings produced by a translation reranker, and reject new strings that have previously been generated. Our technique does not use a hashtable, and instead only stores the hash value for each set of dependencies, which is much more efficient but runs the risk of filtering unique parses due to collisions.

As we combine partial trees to build the derivation, we need to convolve the hash values in a consistent manner. The convolution operator must be *order-independent* as dependencies may be generated in an arbitrary order at different locations in each tree. We use the bitwise exclusive OR ($\oplus$) operation as our convolution operator: when two partial derivations are combined, their hash values are XOR'ed together. XOR is commonly employed in hashing applications for randomly permuting numbers, and it is also order independent: $a \oplus b \equiv b \oplus a$. Using XOR, we enforce a unique hash value constraint in the n-best list of candidates, discarding potential candidates with an identical hash value to any already in the list.

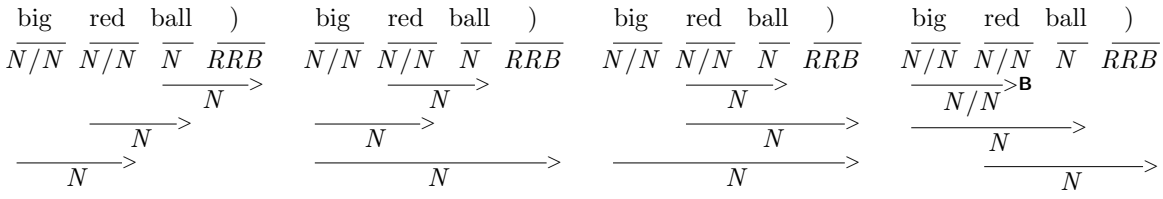

Figure 3: All four derivations have a different syntactic structure, but generate identical dependencies.

	Collisions	Comparisons	%
10-best	300	54861	0.55
50-best	2109	225970	0.93

Table 3: Dependency hash collisions and comparisons over 00 of CCGbank.

3.2 Hashing performance

We evaluate our hashing technique with several experiments. A simple test is to measure the number of collisions that occur, i.e. where two partial trees with different dependencies have the same hash value. We parsed CCGbank 00 with $n = 10$ and $n = 50$ using a 32 bit hash, and exhaustively checked the dependencies of colliding states. We found that less than 1% of comparisons resulted in collisions in both 10-best and 50-best mode, and decided that this was acceptably low for distinguishing duplicates.

We reran the diversity experiments, and verified that every n-best parse for every sentence in CCGbank 00 was unique (see Table 1), corroborating our decision to use hashing alone. On average, there are fewer parses per sentence, showing that hashing is eliminating many equivalent parses for more ambiguous sentences. However, hashing also leads to a near doubling of unique parses in 10-best mode and a 2.3x increase in 50-best mode. Similar results are recorded for the GR diversity (see Table 2), though not every set of GRs is unique due to the many-to-many mapping from CCG dependencies. These results show that hashing prunes away equivalent parses, creating more diversity in the n-best list.

We also evaluate the oracle F-score of the parser using dependency hashing. Our results in Table 4 include a 1.1% increase in 10-best mode and 0.72% in 50-best mode using the new constraints, showing how the diversified parse list contains better candidates for reranking. Our highest oracle F-score was 93.32% in 50-best mode.

Experiment	LP	LR	LF	AF
baseline	87.27	86.41	86.84	84.91
oracle 10-best	91.50	90.49	90.99	89.01
oracle 50-best	93.17	92.04	92.60	90.68
oracle 10-best#	**92.67**	**91.51**	**92.09**	**90.15**
oracle 50-best#	**94.00**	**92.66**	**93.32**	**91.47**

Table 4: Oracle precision, recall, and F-score on gold and auto POS tags for the C&C n-best parser. # denotes the inclusion of dependency hashing.

	Test data	
Training data	no hashing	hashing
no hashing	86.83	86.35
hashing	**87.21**	**87.15**

Table 5: Reranked parser accuracy; labeled F-score using gold POS tags, with and without dependency hashing

3.3 CCG reranking performance

Finally, we implement a discriminative maximum entropy reranker for the n-best C&C parser and evaluate it when using dependency hashing. We reimplement the features described in Charniak and Johnson (2005) and add additional features based on those used in the C&C parser and on features of CCG dependencies. The training data is cross-fold n-best parsed sentences of CCGbank 02-21, and we use the MEGAM optimiser[1] in regression mode to predict the labeled F-score of each n-best candidate parse.

Our experiments rerank the top 10-best parses and use four configurations: with and without dependency hashing for generating the training and test data for the reranker. Table 5 shows that labeled F-score improves substantially when dependency hashing is used to create reranker training data. There is a 0.4% improvement using no hashing at test, and a 0.8% improvement using hashing

[1] http://hal3.name/megam

at test, showing that more diverse training data creates a better reranker. The results of 87.21% without hashing at test and 87.15% using hashing at test are statistically indistinguishable from one other; though we would expect the latter to perform better.

Our results also show that the reranker performs extremely poorly using diversified test parses and undiversified training parses. There is a 0.5% performance loss in this configuration, from 86.83% to 86.35% F-score. This may be caused by the reranker becoming attuned to selecting between semantically indistinguishable derivations, which are pruned away in the diversified test set.

4 Analysing parser errors

A substantial gap exists between the oracle F-score of our improved n-best parser and other PTB n-best parsers (Charniak and Johnson, 2005). Due to the different evaluation schemes, it is difficult to directly compare these numbers, but whether there is further room for improvement in CCG n-best parsing is an open question. We analyse three main classes of errors in the C&C parser in order to answer this question: grammar error, supertagger error, and model error. Furthermore, insights from this analysis will prove useful in evaluating tradeoffs made in parsers.

Grammar error: the parser implements a subset of the grammar and unary type-changing rules in CCGbank for efficiency, with some rules, such as substitution, omitted for efficiency (Clark and Curran, 2007). This means that, given the correct categories for words in a sentence, the parser may be unable to combine them into a derivation yielding the correct dependencies, or it may not recognise the gold standard category at all.

There is an additional constraint in the parser that only allows two categories to combine if they have been seen to combine in the training data. This *seen rules* constraint is used to reduce the size of the chart and improve parsing speed, at the cost of only permitting category combinations seen in CCGbank 02-21 (Clark and Curran, 2007).

Supertagger error: The supertagger uses a restricted set of 425 categories determined by a frequency cutoff of 10 over the training data (Clark and Curran, 2004b). Words with gold categories that are not in this set cannot be tagged correctly.

The β parameter restricts the categories to within a probability beam, and the tag dictionary restricts the set of categories that can be considered for each word. Supertagger model error occurs when the supertagger can assign a word its correct category, but the statistical model does not assign the correct tag enough probability for it to fall within the β.

Model error: The parser model features may be rich enough to capture certain characteristics of parses, causing it to select a suboptimal parse.

4.1 Subtractive experiments

We develop an oracle methodology to distinguish between grammar, supertagger, and model errors. This is the most comprehensive error analysis of a parsing pipeline in the literature.

First, we supplied gold-standard categories for each word in the sentence. In this experiment the parser only needs to combine the categories correctly to form the gold parse. In our testing over CCGbank 00, the parser scores 99.49% F-score given perfect categories, with 95.61% coverage. Thus, grammar error accounts for about 0.5% of overall parser errors as well as a 4.4% drop in coverage[2]. All results in this section will be compared against this 99.49% result as it removes the grammar error from consideration.

4.2 Supertagger and model error

To determine supertagger and model error, we run the parser on standard settings over CCGbank 00 and examined the chart. If it contains the gold parse, then a model error results if the parser returns any other parser. Otherwise, it is a supertagger or grammar error, where the parser cannot construct the best parse. For each sentence, we found the best parse in the chart by decoding against the gold dependencies. Each partial tree was scored using the formula:

$$score = ncorrect - nbad$$

where $ncorrect$ is the number of dependencies which appear in the gold standard, and $nbad$ is the number of dependencies which do not appear in the gold standard. The top scoring derivation in the tree under this scheme is then returned.

[2]Clark and Curran (2004a) performed a similar experiment with lower accuracy and coverage; our improved numbers are due to changes in the parser.

Experiment	LP	LR	LF	AF	cover	ΔLF	ΔAF
oracle cats	99.72	99.27	99.49	99.49	95.61	0.00	0.00
best in chart -tagdict -seen rules	96.88	94.81	95.84	94.17	99.01	-3.65	-5.32
best in chart -tagdict	96.13	94.72	95.42	93.56	99.37	-4.07	-5.93
best in chart -seen rules	96.10	93.66	94.86	93.35	98.85	-4.63	-6.14
best in chart	95.15	93.50	94.32	92.60	99.16	-5.17	-6.89
baseline	87.27	86.41	86.84	84.91	99.16	-12.65	-14.58

Table 6: Oracle labeled precision, recall, F-score, F-score with auto POS, and coverage over CCGbank 00. -tagdict indicates disabling the tag dictionary, -seen rules indicates disabling the seen rules constraint

β	k	cats/word	sent/sec	LP	LR	LF	AF	cover	ΔLF	ΔAF
gold cats		-	-	99.72	99.27	99.49	-	95.61	0.00	0.00
0.075	20	1.27	40.5	95.46	93.90	94.68	93.07	94.30	-4.81	-6.42
0.03	20	1.43	33.0	96.23	94.87	95.54	94.01	96.03	-3.95	-5.48
0.01	20	1.72	19.1	97.02	95.82	96.42	95.02	96.86	-3.07	-4.47
0.005	20	1.98	10.7	97.26	96.09	96.68	95.32	97.23	-2.81	-4.17
0.001	150	3.57	1.18	98.33	97.37	97.85	96.76	96.13	-1.64	-2.73

Table 7: Category ambiguity, speed, labeled P, R, F-score on gold and auto POS, and coverage over CCGbank 00 for the standard supertagger parameters selecting the best scoring parse against the gold parse in the chart.

We obtain an overall maximum possible F-score for the parser using this scoring formula. The difference between this maximum F-score and the oracle result of 99.49% represents supertagger error (where the supertagger has not provided the correct categories), and the difference to the baseline performance indicates model error (where the parser model has not selected the optimal parse given the current categories). We also try disabling the seen rules constraint to determine its impact on accuracy.

The impact of tag dictionary errors must be neutralised in order to distinguish between the types of supertagger error. To do this, we added the gold category for a word to the set of possible tags considered for that word by the supertagger. This was done for categories that the supertagger could use; categories that were not in the permissible set of 425 categories were not considered. This is an optimistic experiment; removing the tag dictionary entirely would greatly increase the number of categories considered by the supertagger and may dramatically change the tagging results.

Table 6 shows the results of our experiments. The delta columns indicate the difference in labeled F-score to the oracle result, which discounts the grammar error in the parser. We ran the experiment in four configurations: disabling the tag dictionary, disabling the seen rules constraint, and disabling both. There are coverage differences of less than 0.5% that will have a small impact on these results.

The "best in chart" experiment produces a result of 94.32% with gold POS tags and 92.60% with auto POS tags. These numbers are the upper bound of the parser with the supertagger on standard settings. Our result with gold POS tags is statistically identical to the oracle experiment conducted by Auli and Lopez (2011), which exchanged brackets for dependencies in the forest oracle algorithm of Huang (2008). This illustrates the validity of our technique.

A perfect tag dictionary that always contains the gold standard category if it is available results in an upper bound accuracy of 95.42%. This shows that overall supertagger error in the parser is around 5.2%, with roughly 1% attributable to the use of the tag dictionary and the remainder to the supertagger model. The baseline parser is 12.5% worse than the oracle categories result due to model error and supertagger error, so model error accounts for roughly 7.3% of the loss.

Eliminating the seen rules constraint contributes to a 0.5% accuracy improvement over both the standard parser configuration and the -tagdict configuration, at the cost of roughly 0.3% coverage to both. This is of similar magnitude to grammar error; but

Experiment	LF	cover	ΔLF
baseline	86.84	99.16	0.00
auto POS parser	86.57	99.16	-0.27
auto POS super	85.33	99.06	-1.51
auto POS both	84.91	99.06	-1.93

Table 8: Labeled F-score, coverage, and deltas over CCGbank 00 for combinations of gold and auto POS tags.

here accuracy is traded off against coverage.

The results also show that model and supertagger error largely accounts for the remaining oracle accuracy difference between the C&C n-best parser and the Charniak/Collins n-best parsers. The absolute upper bound of the C&C parser is only 1% higher than the oracle 50-best score in Table 4, placing the n-best parser close to its theoretical limit.

4.3 Varying supertagger parameters

We conduct a further experiment to determine the impact of the standard β and k values used in the parser. We reran the "best in chart" configuration, but used each standard β and k value individually rather than backing off to a lower β value to find the maximum score at each individual value.

Table 7 shows that the oracle accuracy improves from 94.68% F-score and 94.30% coverage with $\beta = 0.075, k = 20$ to 97.85% F-score and 96.13% coverage with $\beta = 0.001, k = 150$. At higher β values, accuracy is lost because the correct category is not returned to the parser, while lower β values are more likely to return the correct category. The coverage peaks at the second-lowest value because at lower β values, the number of categories returned means all of the possible derivations cannot be stored in the chart. The back-off approach substantially increases coverage by ensuring that parses that fail at higher β values are retried at lower ones, at the cost of reducing the upper accuracy bound to below that of any individual β.

The speed of the parser varies substantially in this experiment, from 40.5 sents/sec at the first β level to just 1.18 sents/sec at the last. This illustrates the trade-off in using supertagging: the maximum achievable accuracy drops by nearly 5% for parsing speeds that are an order of magnitude faster.

4.4 Gold and automatic POS tags

There is a substantial difference in accuracy between experiments that use gold POS and auto POS tags. Table 6 shows a corresponding drop in upper bound accuracy from 94.32% with gold POS tags to 92.60% with auto POS tags. Both the supertagger and parser use POS tags independently as features, but this result suggests that the bulk of the performance difference comes from the supertagger. To fully identify the error contributions, we ran an experiment where we provide gold POS tags to one of the parser and supertagger, and auto POS tags to the other, and then run the standard evaluation (the oracle experiment will be identical to the "best in chart").

Table 8 shows that supplying the parser with auto POS tags reduces accuracy by 0.27% compared to the baseline parser, while supplying the supertagger with auto POS tags results in a 1.51% decrease. The parser uses more features in a wider context than the supertagger, so it is less affected by POS tag errors.

5 Conclusion

We have described how a mismatch between the way CCG parses are modeled and evaluated caused equivalent parses to be produced in n-best parsing. We eliminate duplicates by hashing dependencies, significantly improving the oracle F-score of CCG n-best parsing by 0.7% to 93.32%, and improving the performance of CCG reranking by up to 0.4%.

We have comprehensively investigated the sources of error in the C&C parser to explain the gap in oracle performance compared with other n-best parsers. We show the impact of techniques that subtly trade off accuracy for speed and coverage. This will allow a better choice of parameters for future applications of parsing in CCG and other lexicalised formalisms.

Acknowledgments

We would like to thank the reviewers for their comments. This work was supported by Australian Research Council Discovery grant DP1097291, the Capital Markets CRC, an Australian Postgraduate Award, and a University of Sydney Vice-Chancellor's Research Scholarship.

References

Michael Auli and Adam Lopez. 2011. Training a Log-Linear Parser with Loss Functions via Softmax-Margin. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP-11)*, pages 333–343. Edinburgh, Scotland, UK.

Forrest Brennan. 2008. *k-best Parsing Algorithms for a Natural Language Parser*. Master's thesis, University of Oxford.

Ted Briscoe and John Carroll. 2006. Evaluating the Accuracy of an Unlexicalized Statistical Parser on the PARC DepBank. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 41–48. Sydney, Australia.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pages 173–180. Ann Arbor, Michigan, USA.

Stephen Clark and James R. Curran. 2004a. Parsing the WSJ Using CCG and Log-Linear Models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 103–110. Barcelona, Spain.

Stephen Clark and James R. Curran. 2004b. The Importance of Supertagging for Wide-Coverage CCG Parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, pages 282–288. Geneva, Switzerland.

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

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building Deep Dependency Structures using a Wide-Coverage CCG Parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 327–334. Philadelphia, Pennsylvania, USA.

Michael Collins. 2000. Discriminative Reranking for Natural Language Parsing. In *Proceedings of the 17th International Conference on Machine Learning (ICML-00)*, pages 175–182. Palo Alto, California, USA.

Jason Eisner. 1996. Efficient Normal-Form Parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 79–86. Santa Cruz, California, USA.

Julia Hockenmaier. 2003. Parsing with Generative Models of Predicate-Argument Structure. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, pages 359–366. Sapporo, Japan.

Julia Hockenmaier. 2006. Creating a CCGbank and a Wide-Coverage CCG Lexicon for German. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-06)*, pages 505–512. Sydney, Australia.

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.

Liang Huang. 2008. Forest Reranking: Discriminative Parsing with Non-Local Features. In *Proceedings of the Human Language Technology Conference at the 45th Annual Meeting of the Association for Computational Linguistics (HLT/ACL-08)*, pages 586–594. Columbus, Ohio.

Liang Huang and David Chiang. 2005. Better k-best Parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology (IWPT-05)*, pages 53–64. Vancouver, British Columbia, Canada.

Liang Huang, Kevin Knight, and Aravind K. Joshi. 2006. Statistical Syntax-Directed Translation with Extended Domain of Locality. In *Proceedings of the 7th Biennial Conference of the Association for Machine Translation in the Americas (AMTA-06)*, pages 66–73. Boston, Massachusetts, USA.

Tracy Holloway King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, pages 1–8. Budapest, Hungary.

Adwait Ratnaparkhi. 1996. A Maximum Entropy Model for Part-of-Speech Tagging. In *Proceedings of the 1996 Conference on Empirical Methods in Natural Language Processing (EMNLP-96)*, pages 133–142. Philadelphia, Pennsylvania, USA.

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

Daniel Tse and James R. Curran. 2010. Chinese CCGbank: extracting CCG derivations from the Penn Chinese Treebank. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING-2010)*, pages 1083–1091. Beijing, China.

Aline Villavicencio. 2002. Learning to Distinguish PP Arguments from Adjuncts. In *Proceedings of the 6th Conference on Natural Language Learning (CoNLL-2002)*, pages 84–90. Taipei, Taiwan.