Improving Combinatory Categorial Grammar Parse Reranking with Dependency Grammar Features

Sunghwan Mac Kim[1] *Dominick Ng*[2]
Mark Johnson[1] *James R. Curran*[2]
(1) Department of Computing, Macquarie University, Sydney, NSW, Australia 2109
(2) School of Information Technologies, University of Sydney, Sydney, NSW, Australia 2006
sunghwan.kim@students.mq.edu.au, dominick.ng@sydney.edu.au,
mark.johnson@mq.edu.au, james.r.curran@sydney.edu.au

ABSTRACT
This paper presents a novel method of improving Combinatory Categorial Grammar (CCG) parsing using features generated from Dependency Grammar (DG) parses and combined using reranking. Different grammar formalisms have different strengths and different parsing models have consequently divergent views of the data. More specifically, dependency parsers are sensitive to linguistic generalisations that differ from the generalisations that the CCG parser is sensitive to, and which the reranker exploits to identify the parse most likely to be correct. We propose DG-derived reranking features, which are obtained by comparing dependencies from the CCG parser with DG dependencies, and demonstrate how they improve the performance of a CCG parser and reranker in a variety of settings. We record a final labeled F-score of 87.93% on section 23 of CCGbank, 0.5% and 0.35% improvements over the base parser (87.43%) and reranker (87.58%), respectively.

KEYWORDS: Combinatory Categorial Grammar (CCG), Dependency Grammar (DG), Reranking, Dependency Grammar-derived features, parsing, syntax.

1 Introduction

Reranking is the process of rescoring an *n*-best list with an external model, and it is an effective method for improving performance in NLP tasks. In parsing, rerankers are able to incorporate arbitrary global features from the entire parse tree that would be intractable in a base parser. More informative features can be considered in reranking as the entire parse tree is available, as opposed to the fragments considered in parsing.

In this paper, we propose a simple method for improving the performance of the C&C Combinatory Categorial Grammar (CCG) parser (Clark and Curran, 2007). We parse sentences using the C&C *n*-best parser and a 1-best dependency grammar (DG) parser, and generate DG-derived features by comparing the extracted dependencies from the C&C parser with the DG dependencies. We then incorporate the DG-derived features into the CCG reranker of Ng et al. (2010) to reorder the *n*-best CCG parses using the external parse information. We experiment with both the Maltparser (Nivre et al., 2007b) and the MSTparser (McDonald et al., 2005) as the DG parser. This is the first cross-formalism parser combination experiment for CCG parsing that we are aware of, combining the features and strengths of two different formalisms together.

Previous work has shown that dependency parsers such as the Maltparser perform better on short-range dependencies (McDonald and Nivre, 2007), whereas the C&C parser deals with long-range dependencies more reliably (Clark et al., 2002; Rimell et al., 2009). Short-range dependency information has also been shown to improve parser accuracy (Chen et al., 2009). We show how our new DG-derived features substantially improve parser performance by 0.35% to 87.93%, and improve the accuracy of the C&C parser on both short and long-range dependencies. These results demonstrate how rerankers can successfully combine diverse features from different formalisms for better parsing accuracy.

2 Background

2.1 Reranking and Cross-formalism Parser Combination

Collins (2000) describes reranking for the Collins (Model 2) parser and defined the general approach that has been used for the task since. Reranker training data is produced by parsing 36,000 sentences from sections 02-21 of the Penn Treebank WSJ data (Marcus et al., 1993) using an *n*-best version of the base parser. The parser model used for this process is trained using cross-validation to ensure that overly optimistic parses are not produced. Global features calculated over the whole tree such as context-free rules, *n*-gram ancestors, parent and grandparent relationships, and lexical heads and the distances between them are extracted from the parses and fed into a boosting-based reranker. Collins reports a final PARSEVAL F-score of 89.75%, a 1.55% improvement compared to the baseline parser.

The oracle F-score (given a perfect reranker that always chooses the best *n*-best parse for a sentence) is used to measure the quality of *n*-best parses. Huang and Chiang (2005) describe efficient *n*-best parsing algorithms that have become widely used in the field, including in the Charniak and Johnson (2005) reranker. This system uses a similar setup to the Collins reranker, but adopts a maximum-entropy model along with additional features, including features for subject-verb agreement, *n*-gram local trees, and right-branching factors. In 50-best mode the parser has an oracle F-score of 96.80%, and the reranker produces a final F-score of 91.40% compared to an 89.70% baseline.

Farkas et al. (2011) rerank BitPar, an unlexicalised generative PCFG parser for German (Schmid,

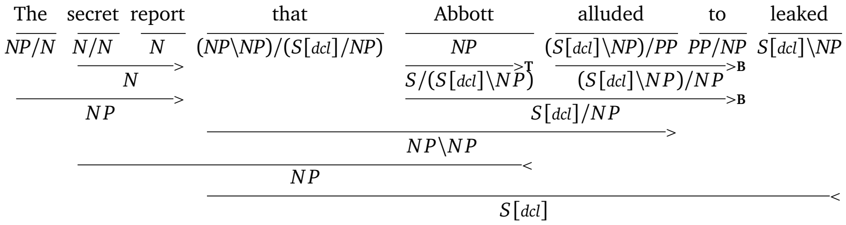

The secret report that Abbott alluded to leaked
NP/N N/N N (NP\NP)/(S[dcl]/NP) NP (S[dcl]\NP)/PP PP/NP S[dcl]\NP
N
NP
S/(S[dcl]\NP)
(S[dcl]\NP)/NP
S[dcl]/NP
NP\NP
NP
S[dcl]

Figure 1: An example CCG derivation using application, composition, type-raising, and unary rules. A long-range dependency is created between *report* and *to*, mediated by *that*.

2004) using dependency grammar features and forest-based rerankers. Bohnet (2010), which is a second order dependency parser, was used to generate parses for feature extraction. Their experimental results show a 0.8% F-score improvement. However, their work was only concerned with extracting additional features from the dependency parses, and does not generate features based on a comparison between the extracted dependency parses and the constituency parses that are being reranked.

Øvrelid et al. (2009) describe a two-stage system where the output of an LFG parser is used to provide features for the Maltparser in English and German. They observe a 0.15% improvement in Maltparser unlabeled attachment scores for English, and 1.81% improvement in German. This work exploits analyses from different formalisms, but it completely retrains the Maltparser with additional features based on a conversion of the LFG analyses to a dependency representation. It also targets improved dependency parsing rather than improved grammar-driven parsing. Sagae et al. (2007) used the output of dependency parser to disambiguate Head-driven Phrase Structure Grammar (HPSG) parses. This work used a single penalty parameter for each mismatch between the DG and HPSG parses that was set via a parameter sweep on held-out data.

2.2 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG; Steedman, 2000) is a lexicalised grammar formalism based on combinatory logic. Lexical categories govern the syntactic behaviour of each word, and generic combinatory rules combine categories together to form an analysis of a sentence.

Atomic categories such as noun phrases (NP) and sentences (S) represent syntactically complete constituents. Complex categories are binary functors of the form A/B or $A\backslash B$, and subcategorize for an argument category B to the right or left respectively to form an A. For example, transitive verbs ($(S\backslash NP)/NP$) subcategorize for an object NP to the right to form a verb phrase $S\backslash NP$, which in turn expects a subject NP to the left to form a sentence S.

The simplest combinatory rules are forward and backward application, where complex categories acquire their outermost argument and return their result. Additional combinators are based on composition and unary category type-changing, increasing the generative power of the formalism and enabling the analysis of phenomena such as *wh*-movement and right-node raising. Figure 1 gives an example CCG derivation using these combinators.

We will use the CCG dependency representation of CCGbank (Hockenmaier and Steedman,

2007) in this work. Each dependency expresses a word-word relationship between a head and a dependent, generated when the assigned categories are combined. Additionally, CCG allows for the production of long-range dependencies mediated by intermediate words in the sentence. This allows a clear representation of function and trace information that would require co-indexation in phrase-structure parses. These dependencies have the following form: $\langle$*to*, PP/NP_1, 1, *report*, $(NP\backslash NP)/(S[dcl]/NP)\rangle$, which includes the head word, its category, the argument slot, argument word, and the mediating category for long-range dependencies.

2.3 CCG parsing

The C&C parser is a fast and accurate wide-coverage CCG parser. It is a two-stage system, where a supertagger assigns probable categories to words in a sentence and the parser combines them using the CKY algorithm. The parser has been found to be particularly accurate at recovering long-range dependencies (Clark et al., 2002; Rimell et al., 2009).

C&C is trained on CCGbank, a conversion of the Penn Treebank WSJ data to CCG derivations and dependencies (Hockenmaier and Steedman, 2007). We use the normal-form model described in Clark and Curran (2007), which models the probability of derivations. We also follow the convention of using section 00 of CCGbank as development data, sections 02-21 as training data, and section 23 for final testing. The standard evaluation metric is labeled dependency recovery, as described by Clark and Hockenmaier (2002).

Clark and Curran (2007) develop a conversion from CCG dependencies to Briscoe and Carroll-style grammatical relations (GRS) (King et al., 2003; Briscoe and Carroll, 2006). GRS provide a useful abstraction as they allow the conflation of many CCG dependencies that are semantically similar but structurally different. For example, since subcategorization information is fully specified in categories, the verb-subject relationship is expressed in many different forms in CCG depending on the transitivity of the verb. In the GR scheme, they map to a general `ncsubj` dependency, echoing the underlying similarity between the CCG dependencies.

Rimell and Clark (2009) adapt the C&C parsing for the biomedical domain, and in the process they developed a mapping from CCG dependencies to Stanford dependencies based on the GR conversion. We generate DG-derived features based on this Stanford dependency output of the C&C parser to maximise the potential overlaps between the representations; this differs from the existing C&C reranking features, which use the CCG dependency format (Ng et al., 2010).

Figure 2 shows the CCG derivation and corresponding Stanford dependencies for the example sentence, *We are about to see if advertising works*, taken from WSJ section 22.

2.4 Dependency parsing

Dependency Grammars (DG) describe the syntactic structure of a sentence in terms of head-dependent relations between words. The set of dependency relations for a sentence forms a dependency tree with a special root head word. Unlike CCG, DG directly model relationships between pairs of words, and do not easily account for mediated long-range dependencies. CCG categories encoded detailed subcategorization information that is not present in DG labels, and the restrictions in combinator application constrain the way CCG derivations can be built, whereas dependency arcs may appear between any pair of words under a DG. Thus, we expect that CCG and DG analyses will provide markedly different insights, despite both producing dependency-style output.

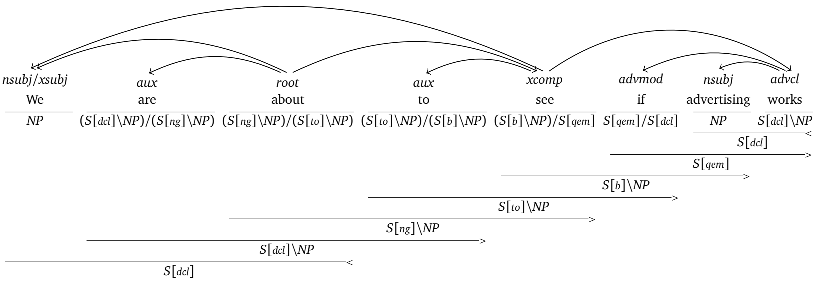


Figure 2: A CCG derivation and the Stanford dependencies produced by applying the Rimell and Clark (2009) conversion on the parse.

2.5 DG Representation Schemes

We experimented with four different dependency schemes; for each scheme, we retrained the Maltparser and the MSTparser over the extracted dependencies from the Penn Treebank WSJ data. 20-fold cross-validation was used to generate the parses corresponding to the reranker training data (sections 02-21); sections 02-21 was used to create a model for use at test time.

CoNLL: The CoNLL DG was used in the CoNLL 2007 dependency parsing shared task (Nivre et al., 2007a). Penn2Malt, a publicly available conversion utility[1], was used to generate CoNLL dependencies for our experiments. In contrast to other grammars used in this paper, this dependency scheme contains only unlabeled word-word arcs.

Stanford: de Marneffe and Manning (2008) introduced the dependency scheme used in the Stanford parser[2]. We used the Stanford parser's built-in converter to transform Penn Treebank trees into dependencies. The Stanford scheme has different variants; for this work we use the basic projective tree schema.

LTH: The LTH dependency scheme was developed with the aim of making better use of the linguistic information present in the Penn Treebank from version II onwards (Johansson and Nugues, 2007). We generated these dependencies using the LTH converter[3] over the NP-bracketed version of the Penn Treebank described by Vadas and Curran (2007). The converter was configured to produce a functional rather than lexical DG.

Fanse: Another conversion of the Penn Treebank with more fine-grained labels was presented in Tratz and Hovy (2011). The Fanse scheme is linguistically rich, featuring both non-projective dependencies and shallow semantic interpretation in its analyses. We used the freely available converter[4], which also requires the Vadas and Curran (2007) NP-bracketed Penn Treebank.

Figures 3 and 4 demonstrate some of the differences between the four dependency schemes. For instance, auxiliaries take the lexical verb as a dependent in all schemes except for Stanford,

[1]http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html
[2]http://nlp.stanford.edu/software/lex-parser.shtml
[3]http://nlp.cs.lth.se/software/treebank_converter/
[4]http://www.isi.edu/publications/licensed-sw/fanseparser/

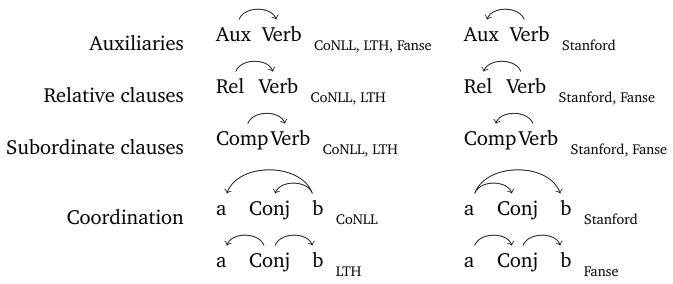


Figure 3: Analyses of auxiliaries, relative/subordinate clauses and coordination in the DG schemes.

where the lexical verb is the head of a *VP*. The characteristics of each scheme mean that each one produces an analysis that is quite different to the others as well as to CCG; by investigating a variety of schemes we hope to identify characteristics which are useful in our cross-formalism experiment.

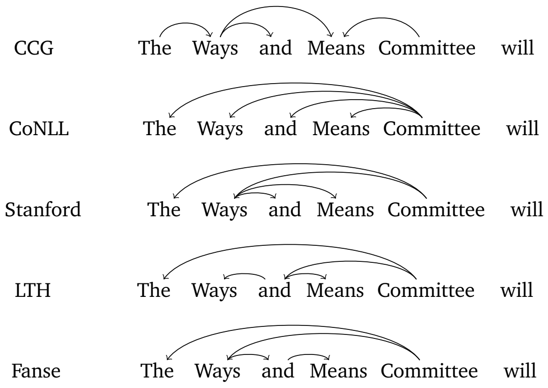


Figure 4: Example of divergence on the interpretation of the coordination by each scheme.

2.6 Maltparser and MSTparser

DG schemes	Maltparser	MSTparser
CoNLL	**90.46**	88.77
Stanford	89.82	87.27
LTH	84.54	86.67
Fanse	89.96	**89.61**

Table 1: Unlabeled Attachment Scores for each scheme over WSJ section 22.

In this work we use the Maltparser, a transition-based dependency parser (Nivre et al., 2007b), and the MSTparser, a graph-based dependency parser (McDonald et al., 2005). The Maltparser uses an incremental shift-reduce algorithm, with actions guided by a classifier trained over parse history information. In contrast, the MSTparser builds a weighted graph for sentences, and finds the parse corresponding to the maximum spanning tree of the graph.

Table 1 shows the Unlabeled Attachment Scores (UASs) over WSJ section 22 for the Maltparser and the MSTparser with respect to four DG schemes[5]. The Maltparser has the highest UAS (90.46%) with the CoNLL DG, while the MSTparser performs best (89.61%) using the Fanse DG. Both parsers perform the worst with the LTH DG and by a substantial margin: 6% F-score for the Maltparser and 3% for the MSTparser compared with the best result. These results lead us to expect that features generated from Maltparser output will perform better than those from the MSTparser, LTH scheme notwithstanding.

3 CCG reranking

We follow the CCG reranker implementation described in Ng et al. (2010) and use the *n*-best C&C parser described in Brennan (2008); Ng and Curran (2012). This reranker is inspired by Charniak and Johnson (2005), with many new features designed to address specifics of the CCG formalism and evaluation process. For each *n*-best parse, the reranker uses a regression model to predict its expected F-score, and chooses the model with the highest predicted score. The log score and rank assigned to each derivation by the parser were encoded as core features in the reranker, and here we briefly summarise the other feature groups:

Tree Topology features describe the overall shape of the parse tree, to capture the fact that English generally favours right-branching parse trees, with heavy constituents generally occurring in the sentence-final position.

Local Context features represent fragments of the tree as well as layers of vertical and horizontal context that are difficult to encode in the parser model.

Argument-Adjunct features represent different attachment points for arguments and adjuncts and their wider context. Incorrect argument-adjunct distinctions can cause multiple CCG dependency errors due to the subcategorization information encoded in CCG categories.

Grammar-based features encode combinator sequences or combinations that may indicate an overly complicated or undesirable derivation. Additionally, these features encode the actual dependencies as these are the target of the evaluation.

C&C features from the parser are also incorporated as described in Clark and Curran (2007). These features encode combinations of word-category, word-POS, root-word, CCG rule, distance, and dependency information.

4 DG-derived Features

This section describes the DG-derived features. For convenience, we refer to the converted Stanford dependency output of an *n*-best CCG parse as the CCG *Dependency Parse* (CDP), as opposed to the *Malt/MST Dependency Parse* (MDP).

Our DG features are designed to capture the desirable and undesirable characteristics of the CDP and MDP, as well as the ways in which dependencies match and mismatch between the two. The dependencies from each *n*-best CDP are compared pairwise with the dependencies from the

[5]20-fold cross-validation on sections 02-21 is used to create reranker training data.

1-best MDP for the corresponding sentence. *Matching* dependencies are those where the head and dependent are the same in the CDP and MDP. These dependencies also have a directionality component: whether the match occurs in the *same* direction (head and dependent are the same in CDP and MDP), or in the *reverse* direction (head and dependent are in opposition in the CDP and MDP). *Mismatching* dependencies are classified as being a head-dependent pair existing only in the CDP, or only in the MDP.

Our intuition is that the reranker should learn to prefer or disprefer particular properties of matching and mismatching dependencies. For example, it may learn that a particular CCG dependency is usually expressed in the same or opposite direction in the dependency parse. It may learn that a particular dependency is always expected to be mismatching, or that particular heads tend to take the same pairs of dependents in both parses.

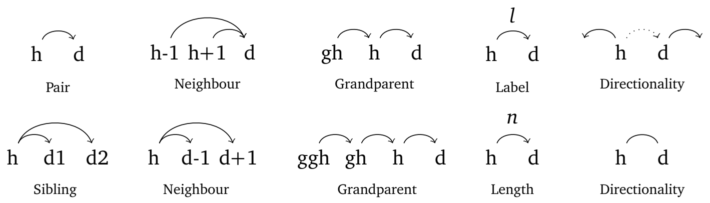


Figure 5: Dependency representations for each feature group. The letters ggh, gh, h, d refer to great grandparent head, grandparent head, head and dependent respectively.

For each matching dependency, binary indicator features were generated based on our feature templates (depicted in Figure 5). These features represent fragments of one or more dependency arcs that the reranker learns to favour or disprefer. Each feature includes components specified by the template, a directionality marker, and all four combinations of the word and POS tag for the head and dependent. Some of our templates also generate additional features for each mismatching dependency, conjoined with a label indicating whether the dependency existed only in the CDP or the MDP.

This approach differs from that of Sagae et al. (2007) since our reranker learns a separate penalty parameter for each combination of DG and CCG constructions as a feature of our regularised MaxEnt reranker model; these weights are learnt as part of the reranker training procedure. This enables our reranker to learn which DG constructions are most reliable and informative for CCG parsing, and which DG constructions should be ignored.

Following are descriptions of our DG-derived feature templates, which correspond to various dependency relations shown in Figure 5:

Pair-dependency features encode the head-dependent pair and a flag indicating a matching or mismatching dependency.

Sibling-dependency features encode matching sets of heads and pairs of dependents between the CDP and MDP. Multiple features are generated for each additional pair of matching dependents per head.

Neighbour-dependency features encode the linear context of heads and dependents in a sentence. For each matching dependency, the head is encoded with the word and POS tag in turn of words immediately to the left and right of its dependent. This procedure is repeated for the dependent with the neighbours of its head.

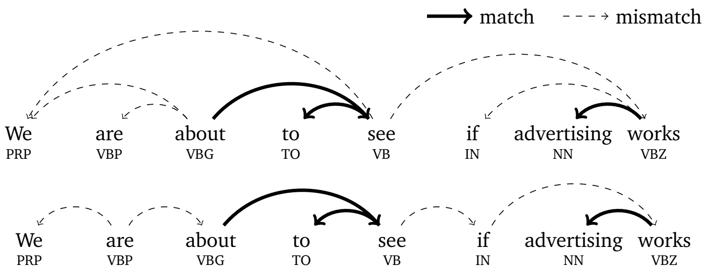
Grandparent-dependency features encode matching relationships of a great-grandparent head, grandparent head, head, and dependent between the CDP and MDP.

We also developed features that included further arc-level information from the dependency parses. These included:

Label-dependency features mimic the pair dependency features, but also include the label assigned by the dependency parser to the arc. This feature was not used for the CoNLL scheme (as this scheme does not include labels), but was active for each of the others.

Length-dependency features encode matching dependencies conjoined with the bucketed length of the dependency with respect to intervening tokens between head and dependent. The bucket intervals were set at 1, 2, 5, and 8 based on an analysis of typical lengths.

Directionality-dependency features consider, given a matching dependency, the additional matching dependencies that the head or dependent are part of. We term the dependencies in this set that are headed by the head or dependent as *out arcs*, and all others as *in arcs*. This feature template encodes the number of out arcs for each matching dependency, as well as a real value feature encoding the ratio of out arcs to in arcs.



Match: (about, see) (see, to) (works, advertising)
Mismatch^{CDP}: (about, are) (about, We) (see, We) (see, works) (works, if)
Mismatch^{MDP}: (are, We) (are, about) (see, if) (if, works)

Figure 6: Matching and mismatching pair dependencies between a CCG parse (top) and CoNLL parse (bottom) for the sentence *We are about to see if advertising works*.

Figure 6 illustrates matching and mismatching pair dependencies for the sentence shown in Figure 2, with dependencies involving punctuation ignored. There are three matching dependencies, five mismatching dependencies from to the CCG parse, and four mismatching dependencies from the DG parse. Table 2 lists examples of the pair-dependency features generated from these parses.

5 Results

We ran two classes of reranker experiments for each dependency scheme: one using gold-standard DG dependencies that were directly converted from the treebank data, and one using

Dependency	Features
(about, see)	`match:about:see, match:about:VB, match:VBG:see, match:VBG:VB`
(see, We)	`nomatch-ccg:see:We, nomatch-ccg:VB:We, nomatch-ccg:see:PRP, nomatch-ccg:VB:PRP`
(see, if)	`nomatch-malt:see:if, nomatch-malt:VB:if, nomatch-malt:see:IN, nomatch-malt:VB:IN`

Table 2: The generation process for pair-dependency features. Each feature template follows a similar pattern.

parser-predicted DG dependencies. The gold experiment allowed us to investigate the upper performance bound of our reranking technique and of our DG-derived features. We evaluate using the standard CCG dependency recovery metric over section 00 of CCGbank.

We use the reranker settings that Ng et al. (2010) found to provide best performance: regression learning, 10-best mode, and no feature pruning. We use the same experimental settings reported in Nivre et al. (2010) and McDonald et al. (2005) for the Maltparser and MSTparser respectively. This means that both parsers will produce a projective dependency tree for each scheme that we experimented with.

5.1 Overall Comparison

section 00 (dev)		LP	LR	LF
Baselines	C&C normal '07	87.27	86.41	86.84
	Reranker '10	87.57	86.69	87.13
Gold	CoNLL features	89.17	88.21	88.69
	Stanford features	88.97	88.06	88.51
	LTH features	88.95	88.01	88.48
	Fanse features	**89.61**	**88.72**	**89.16**
Malt Predicted	CoNLL features	87.74	86.85	87.29
	Stanford features	87.80	86.90	87.35
	LTH features	87.43	86.50	86.96
	Fanse features	**87.82**	**86.93**	**87.37**
MST Predicted	CoNLL features	**87.65**	**86.77**	**87.21**
	Stanford features	87.60	86.71	87.15
	LTH features	87.58	86.69	87.14
	Fanse features	87.61	86.72	87.17

Table 3: Parsing performance for the four DG schemes in gold and predicted configurations over section 00 of CCGbank

Table 3 records the labeled precision (LP), recall (LR) and F-score (LF) per system over section 00 of CCGbank. We compare our results to that of the C&C'07 baseline and the Reranker '10 baseline of Ng et al. (2010); the latter uses only the features defined in Ng et al. (2010), whereas our work includes the DG-derived features previous described. Each dependency scheme was tested in turn with the same feature set.

The gold results show that performance improvements of over 2% F-score are possible with perfect DG-derived features. Each of the DG schemes performs similarly, with the exception of the Fanse scheme, which provides roughly 0.5% higher F-score than the others. The Fanse scheme is generated with the NP-bracketed version of the Penn Treebank, and also incorporates deeper linguistic analysis than the other schemes. However, it is interesting that the LTH dependencies generated from the same enriched corpus have the lowest upper bound.

The Maltparser-predicted results show that, with the exception of the LTH DG, the DG-derived features perform roughly on par with each other and generally better than the Reranker '10. However, the LTH features perform substantially worse than the others and also worse than Reranker '10; this corresponds with the Maltparser performing worst with respect to unlabeled attachment on that scheme. In contrast, the MSTparser feature results are each indistinguishable from one another, despite the very different performance of the baseline with respect to each scheme. The MSTparser results are also indistinguishable from the Reranker '10 system; this shows that the MSTparser does not produce enough useful variations compared with CCG for our procedure to work.

5.2 Isolation experiments

We took our best performing systems, which used the Fanse and CoNLL-based features for the Maltparser and MSTparser respectively, and investigated the individual impact of the DG-derived features. We did this by running the reranker with DG-derived features only over section 00 of CCGbank, and comparing our results with those of Reranker '10 (CCG features only) and the full system. Table 4 summarises the results; DG denotes our new features, and CCG denotes the CCG features.

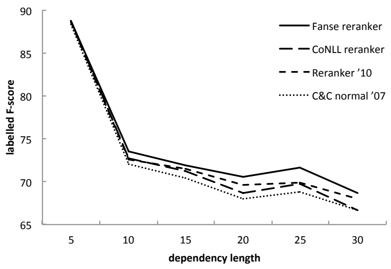
section 00 (dev)		LP	LR	LF
Reranker '10	CCG	87.57	86.69	87.13
Gold	Fanse DG	89.43	88.44	88.93
	CCG +Fanse DG	**89.61**	**88.72**	**89.16**
Malt Predicted	Fanse DG	86.79	85.79	86.29
	CCG +Fanse DG	**87.82**	**86.93**	**87.37**
MST Predicted	CoNLL DG	85.94	84.80	85.36
	CCG +CoNLL DG	**87.65**	**86.77**	**87.21**

Table 4: A comparison of DG-derived features in isolation and in combination with CCG reranker features over section 00 of CCGbank.

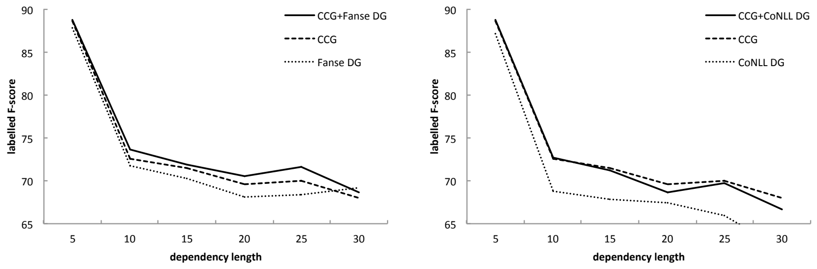
The gold results show that the DG-derived features perform strongly in isolation and outperform Reranker '10. However, performance is much worse using automatic DG parses compared to Reranker '10; F-score drops from 88.93% to 86.29% for the Maltparser-predicted experiment, and to 85.35% for MSTparser-predicted. These experiments are simply the result of removing the CCG features from the best performing system, and show that automatically produced DG features in isolation are harmful for reranking. However, the abstraction provided by automatic DG features prove useful in combination with CCG features and lead to a performance improvement.

5.3 Dependency lengths

Figure 7-(a) plots the labeled F-scores with respect to dependency length (number of words between the head and dependent) on section 00 for the best Fanse and CoNLL features. Our



(a) System comparison



(b) Feature comparison

Figure 7: Labeled F-score performance by bucketed dependency lengths for the Maltparser-predicted Fanse and MSTparser-predicted CoNLL experiments. Test conducted over CCGbank section 00.

new DG-derived features improve the Fanse performance across all dependency lengths compared to the C&C baseline and Reranker '10. In particular, we have improvements of 2.86% and 1.67% over the C&C baseline and Reranker '10 respectively when the dependency length is between 21 and 25, though the number of these dependencies is relatively small. In contrast, the MSTparser-predicted CoNLL features perform worse than the Reranker '10 as the dependency length grows though even though the CoNLL features perform better than the C&C'07 baseline for all dependency lengths.

We repeated the dependency length analysis with DG-derived features in isolation, as per the previous section. As can be seen from Figure 7-(b), Fanse DG-derived features in isolation perform poorly, while combining CCG and DG features gives an overall performance improvement over CCG features alone for all dependency lengths. Our CCG +Fanse DG reranker performs well for dependency lengths 21-25, with an F-score difference of 1.67% compared to Reranker '10. However, the F-score of CCG +CoNLL DG starts falling below the performance of CCG features beyond the dependency length 10.

These results all show a similar pattern, where F-score is very high for short dependencies, drops sharply up to length 6-10 words, and levels out for longer lengths. Interestingly, we notice that the performance using only Fanse DG-derived features slightly increases as the dependency length grows beyond 15. For dependencies of length 25 and longer, the Fanse features in isolation actually outperform CCG features in isolation. Additionally, the performance improvement for the CCG +Fanse features over Reranker '10 is more substantial at longer dependency lengths – contrary to our initial expectations given the relative strengths of the parsers we used. One possible reason may be due to the generation of the Fanse scheme over the NP-enriched Penn Treebank. In contrast, the C&C parser was trained over a non-enriched version of CCGbank where all the noun phrases are right-branching. More errors may have crept into the noun phrase results as they typically contain short dependencies.

5.4 Subtractive feature analysis

section 00 (dev)		LP	LR	LF
Gold	CCG +Fanse DG	89.61	88.72	89.16
	-Pair/Neighbour	89.49	88.61	89.05
	-Sibling	89.64	88.76	89.20
	-Grandparent	89.58	88.71	89.14
	-Label	89.53	88.63	89.07
	-Length	**89.13**	**88.25**	**88.69**
	-Directionality	**89.18**	**88.31**	**88.74**
Malt Predicted	CCG +Fanse DG	87.82	86.93	87.37
	-Pair/Neighbour	87.66	86.78	87.22
	-Sibling	87.69	86.80	87.24
	-Grandparent	87.72	86.85	87.28
	-Label	87.67	86.79	87.23
	-Length	87.68	86.82	87.25
	-Directionality	87.69	86.82	87.25
MST Predicted	CCG +CoNLL DG	87.65	86.77	87.21
	-Pair/Neighbour	87.56	86.67	87.12
	-Sibling	87.53	86.65	87.09
	-Grandparent	87.58	86.70	87.14
	-Length	87.56	86.69	87.12
	-Directionality	87.76	86.84	87.30

Table 5: Subtractive analysis of each DG feature for the Fanse and CoNLL schemes on CCGbank section 00. Bolded rows indicate the most substantial performance drops.

We performed a subtractive analysis to investigate the individual contribution of each feature type. We can see in Table 5 that different features are important for the gold and predicted experiments. Removing the length or directionality-dependency features causes a substantial performance decrease in the gold-standard experiment. This seems to suggest that certain dependencies between words will only exist with a certain length between head and dependent, and that words also have a relatively predictable number of incoming and outgoing arcs in the gold standard.

In contrast, the removal of no individual feature causes a significant change in F-score in either the Maltparser-predicted or MSTparser-predicted experiments. The introduction of parse errors has reduced the reliance of the reranker on length and directionality features, and caused it to smooth out the weights to other features. In this setting it is the combination of small contributions from many features that is important.

5.5 Final results

We used the Fanse and CoNLL features, as these performed best on the development data for the Maltparser and MSTparser respectively, for a single run on the final test set of section 23 of CCGbank. Table 7 shows that the Fanse features with the Maltparser improve F-score relative to the C&C baseline and Reranker '10 systems. In particular, our final result of 87.93% F-score is a 0.5% improvement over the baseline parser, and a 0.35% gain over Reranker '10. These results are significant at $p < 0.05$, as tested using Bikel's approximate randomisation procedure [6].

section 23 (test)	LP	LR	LF
C&C normal '07	87.81	87.06	87.43
Reranker '10 (CCG)	87.98	87.18	87.58
Fanse CCG +DG features (Maltparser)	**88.32**	**87.54**	**87.93**
CoNLL CCG +DG features (MSTparser)	88.02	87.20	87.61

Table 7: Final test results for the best DG parser-predicted features over section 23 of CCGbank.

Conclusion

In this paper we proposed new DG-derived features, generated by a dependency parser and incorporated into a CCG parser using reranking. We observe significant performance improvements using the DG-derived features from the Maltparser, and also show that there remains substantial potential in the DG-derived features with improved dependency parsing.

The LTH and Fanse dependency schemes were created from the NP-bracketed Penn Treebank of Vadas and Curran (2007). While the Fanse features performed the best in our experiments with the Maltparser, the LTH features performed poorly, further work should experiment with the standard Penn Treebank as the basis for conversion, as well as generating the CoNLL and Stanford schemes with the enriched corpus.

We plan to develop features to address more specific linguistic phenomena such as coordination and prepositional phrase attachment. The dependency schemes each represent these phenomena differently (see Section 2.5), and they are a particular issue in parsing. New features that compare the way different schemes represent these phenomena may allow us to better represent and reproduce them in parsing.

Our work focused on English parsing and we used the Maltparser and MSTparser independently of one another to generate features. It would be interesting to examine the effect of our features on parsing and reranking in different languages, as well as developing new features that compare the output of both dependency parsers. Additionally, we did not use higher order features for the Maltparser and MSTparser. Since these features are not present in the C&C parser, further work should explore whether enabling these features helps reranking.

We have developed a technique for incorporating parse information from one formalism as features for reranking another. This allows us to exploit the strengths of each formalism in a flexible framework with arbitrarily complex features.

[6] based on Dan Bikel's script at `http://www.cis.upenn.edu/~dbikel/software.html#comparator`

Acknowledgments

We would like to thank Matthew Honnibal for his valuable feedback on this work and thank the anonymous reviewers for their helpful comments. This research has been supported by a Macquarie University Research Excellence Scholarship, a Faculty of Science Postgraduate Research Fund and an Australian Research Council DP110102593. The second author was supported by an Australian Postgraduate Award and a Vice-Chancellor's Research Scholarship.

References

Bohnet, B. (2010). Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China.

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

Briscoe, T. and Carroll, J. (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.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine *n*-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 173–180, Ann Arbor, Michigan.

Chen, W., Kawahara, D., Uchimoto, K., Zhang, Y., and Isahara, H. (2009). Using short dependency relations from auto-parsed data for chinese dependency parsing. *ACM Transactions on Asian Language Information Processing (TALIP)*, 8(3):10.

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

Clark, S. and Hockenmaier, J. (2002). Evaluating a Wide-Coverage CCG Parser. In *Proceedings of the Beyond PARSEVAL Workshop at the 3rd International Conference on Language Resources and Evaluation (LREC-02)*, pages 60–66, Las Palmas, Canary Islands, Spain.

Clark, S., Hockenmaier, J., and Steedman, M. (2002). Building deep dependency structures using a wide-coverage CCG parser. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 327–334, Philadelphia, Pennsylvania, USA.

Collins, M. (2000). Discriminative reranking for natural language parsing. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML 2000)*, pages 175–182, Stanford, California.

de Marneffe, M.-C. and Manning, C. D. (2008). The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK.

Farkas, R., Bohnet, B., and Schmid, H. (2011). Features for phrase-structure reranking from dependency parses. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT-11)*, pages 209–214.

Hockenmaier, J. and Steedman, M. (2007). CCGbank: a corpus of CCG derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, pages 355–396.

Huang, L. and Chiang, D. (2005). Better k-best Parsing. In *Proceedings of the 9th International Workshop on Parsing Technology (IWPT-05)*, pages 53–64, Vancouver, British Columbia, Canada.

Johansson, R. and Nugues, P. (2007). Extended constituent-to-dependency conversion for english. In *Proc. of the 16th Nordic Conference on Computational Linguistics (NODALIDA)*.

King, T. H., Crouch, R., Riezler, S., Dalrymple, M., and Kaplan, R. M. (2003). The PARC 700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, pages 1–8, Budapest, Hungary.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

McDonald, R., Crammer, K., and Pereira, F. (2005). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

McDonald, R. and Nivre, J. (2007). Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.

Ng, D. and Curran, J. R. (2012). Dependency Hashing for n-best CCG Parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL-12)*, pages 497–505, Jeju Island, Korea.

Ng, D., Honnibal, M., and Curran, J. R. (2010). Reranking a wide-coverage CCG parser. In *Proceedings of the Australasian Language Technology Association Workshop 2010*, pages 90–98, Melbourne, Australia.

Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., and Yuret, D. (2007a). The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007b). Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

Nivre, J., Rimell, L., McDonald, R., and Gómez Rodríguez, C. (2010). Evaluation of dependency parsers on unbounded dependencies. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 833–841, Beijing, China.

Øvrelid, L., Kuhn, J., and Spreyer, K. (2009). Cross-framework parser stacking for data-driven dependency parsing. *Traitement Automatique des Langues (TAL) Special Issue on Machine Learning for NLP*, 50(3):109–138.

Rimell, L. and Clark, S. (2009). Porting a lexicalized-grammar parser to the biomedical domain. *Journal of Biomedical Informatics*, 42(5):852–865.

Rimell, L., Clark, S., and Steedman, M. (2009). Unbounded dependency recovery for parser evaluation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 813–821, Singapore.

Sagae, K., Miyao, Y., and Tsujii, J. (2007). Hpsg parsing with shallow dependency constraints. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 624–631, Prague, Czech Republic. Association for Computational Linguistics.

Schmid, H. (2004). Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proceedings of Coling 2004*, pages 162–168, Geneva, Switzerland.

Steedman, M. (2000). *The Syntactic Process*. MIT Press, Cambridge, MA.

Tratz, S. and Hovy, E. (2011). A fast, accurate, non-projective, semantically-enriched parser. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1257–1268, Edinburgh, Scotland, UK.

Vadas, D. and Curran, J. (2007). Adding noun phrase structure to the Penn Treebank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 240–247, Prague, Czech Republic.