Shift-Reduce CCG Parsing

Yue Zhang
University of Cambridge
Computer Laboratory
yue.zhang@cl.cam.ac.uk

Stephen Clark
University of Cambridge
Computer Laboratory
stephen.clark@cl.cam.ac.uk

Abstract

CCGs are directly compatible with binary-branching bottom-up parsing algorithms, in particular CKY and shift-reduce algorithms. While the chart-based approach has been the dominant approach for CCG, the shift-reduce method has been little explored. In this paper, we develop a shift-reduce CCG parser using a discriminative model and beam search, and compare its strengths and weaknesses with the chart-based C&C parser. We study different errors made by the two parsers, and show that the shift-reduce parser gives competitive accuracies compared to C&C. Considering our use of a small beam, and given the high ambiguity levels in an automatically-extracted grammar and the amount of information in the CCG lexical categories which form the shift actions, this is a surprising result.

1 Introduction

Combinatory Categorial Grammar (CCG; Steedman (2000)) is a lexicalised theory of grammar which has been successfully applied to a range of problems in NLP, including treebank creation (Hockenmaier and Steedman, 2007), syntactic parsing (Hockenmaier, 2003; Clark and Curran, 2007), logical form construction (Bos et al., 2004) and surface realization (White and Rajkumar, 2009). From a parsing perspective, the C&C parser (Clark and Curran, 2007) has been shown to be competitive with state-of-the-art statistical parsers on a variety of test suites, including those consisting of grammatical relations (Clark and Curran, 2007), Penn Treebank phrase-structure trees (Clark and Curran, 2009), and unbounded dependencies (Rimell et al., 2009).

The binary branching nature of CCG means that it is naturally compatible with bottom-up parsing algorithms such as shift-reduce and CKY (Ades and Steedman, 1982; Steedman, 2000). However, the parsing work by Clark and Curran (2007), and also Hockenmaier (2003) and Fowler and Penn (2010), has only considered chart-parsing. In this paper we fill a gap in the CCG literature by developing a shift-reduce parser for CCG.

Shift-reduce parsers have become popular for dependency parsing, building on the initial work of Yamada and Matsumoto (2003) and Nivre and Scholz (2004). One advantage of shift-reduce parsers is that the scoring model can be defined over actions, allowing highly efficient parsing by using a greedy algorithm in which the highest scoring action (or a small number of possible actions) is taken at each step. In addition, high accuracy can be maintained by using a model which utilises a rich set of features for making each local decision (Nivre et al., 2006).

Following recent work applying global discriminative models to large-scale structured prediction problems (Collins and Roark, 2004; Miyao and Tsujii, 2005; Clark and Curran, 2007; Finkel et al., 2008), we build our shift-reduce parser using a global linear model, and compare it with the chart-based C&C parser. Using standard development and test sets from CCGbank, our shift-reduce parser gives a labeled F-measure of 85.53%, which is competitive with the 85.45% F-measure of the C&C parser on recovery of predicate-argument dependencies from CCGbank. Hence our work shows that

Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics, pages 683–692,
Portland, Oregon, June 19-24, 2011.

transition-based parsing can be successfully applied to CCG, improving on earlier attempts such as Hassan et al. (2008). Detailed analysis shows that our shift-reduce parser yields a higher precision, lower recall and higher F-score on most of the common CCG dependency types compared to C&C.

One advantage of the shift-reduce parser is that it easily handles sentences for which it is difficult to find a spanning analysis, which can happen with CCG because the lexical categories at the leaves of a derivation place strong contraints on the set of possible derivations, and the supertagger which provides the lexical categories sometimes makes mistakes. Unlike the C&C parser, the shift-reduce parser naturally produces fragmentary analyses when appropriate (Nivre et al., 2006), and can produce sensible local structures even when a full spanning analysis cannot be found.[1]

Finally, considering this work in the wider parsing context, it provides an interesting comparison between heuristic beam search using a rich set of features, and optimal dynamic programming search where the feature range is restricted. We are able to perform this comparison because the use of the CCG supertagger means that the C&C parser is able to build the complete chart, from which it can find the optimal derivation, with no pruning whatsoever at the parsing stage. In contrast, the shift-reduce parser uses a simple beam search with a relatively small beam. Perhaps surprisingly, given the ambiguity levels in an automatically-extracted grammar, and the amount of information in the CCG lexical categories which form the shift actions, the shift-reduce parser using heuristic beam search is able to outperform the chart-based parser.

2 CCG Parsing

CCG, and the application of CCG to wide-coverage parsing, is described in detail elsewhere (Steedman, 2000; Hockenmaier, 2003; Clark and Curran, 2007). Here we provide only a short description.

During CCG parsing, adjacent categories are combined using CCG's combinatory rules. For example, a verb phrase in English ($S\backslash NP$) can combine with an NP to its left using function application:

$$NP \;\; S\backslash NP \Rightarrow S$$

Categories can also combine using function composition, allowing the combination of "may" $((S\backslash NP)/(S\backslash NP))$ and "like" $((S\backslash NP)/NP)$ in coordination examples such as "John may like but may detest Mary":

$$(S\backslash NP)/(S\backslash NP) \;\; (S\backslash NP)/NP \Rightarrow (S\backslash NP)/NP$$

In addition to binary rules, such as function application and composition, there are also unary rules which operate on a single category in order to change its type. For example, forward type-raising can change a subject NP into a complex category looking to the right for a verb phrase:

$$NP \Rightarrow S/(S\backslash NP)$$

An example CCG derivation is given in Section 3.

The resource used for building wide-coverage CCG parsers of English is CCGbank (Hockenmaier and Steedman, 2007), a version of the Penn Treebank in which each phrase-structure tree has been transformed into a normal-form CCG derivation. There are two ways to extract a grammar from this resource. One approach is to extract a lexicon, i.e. a mapping from words to sets of lexical categories, and then manually define the combinatory rule schemas, such as functional application and composition, which combine the categories together. The derivations in the treebank are then used to provide training data for the statistical disambiguation model. This is the method used in the C&C parser.[2]

The second approach is to read the complete grammar from the derivations, by extracting combinatory rule *instances* from the local trees consisting of a parent category and one or two child categories, and applying only those instances during parsing. (These rule instances also include rules to deal with punctuation and unary type-changing rules, in addition to instances of the combinatory rule schemas.) This is the method used by Hockenmaier (2003) and is the method we adopt in this paper.

Fowler and Penn (2010) demonstrate that the second extraction method results in a context-free approximation to the grammar resulting from the first

[1] See e.g. Riezler et al. (2002) and Zhang et al. (2007) for chart-based parsers which can produce fragmentary analyses.

[2] Although the C&C default mode applies a restriction for efficiency reasons in which only rule instances seen in CCGbank can be applied, making the grammar of the second type.

method, which has the potential to produce a mildly-context sensitive grammar (given the existence of certain combinatory rules) (Weir, 1988). However, it is important to note that the advantages of CCG, in particular the tight relationship between syntax and semantic interpretation, are still maintained with the second approach, as Fowler and Penn (2010) argue.

3 The Shift-reduce CCG Parser

Given an input sentence, our parser uses a stack of partial derivations, a queue of incoming words, and a series of actions—derived from the rule instances in CCGbank—to build a derivation tree. Following Clark and Curran (2007), we assume that each input word has been assigned a POS-tag (from the Penn Treebank tagset) and a set of CCG lexical categories. We use the same maximum entropy POS-tagger and supertagger as the C&C parser. The derivation tree can be transformed into CCG dependencies or grammatical relations by a post-processing step, which essentially runs the C&C parser deterministically over the derivation, interpreting the derivation and generating the required output.

The configuration of the parser, at each step of the parsing process, is shown in part (a) of Figure 1, where the stack holds the partial derivation trees that have been built, and the queue contains the incoming words that have not been processed. In the figure, $\mathtt{S}_{(\mathtt{H})}$ represents a category S on the stack with head word H, while $\mathtt{Q}_i$ represents a word in the incoming queue.

The set of action types used by the parser is as follows: {SHIFT, COMBINE, UNARY, FINISH}. Each action type represents a set of possible actions available to the parser at each step in the process.

The SHIFT-X action pushes the next incoming word onto the stack, and assigns the lexical category X to the word (Figure 1(b)). The label X can be any lexical category from the set assigned to the word being shifted by the supertagger. Hence the shift action performs lexical category disambiguation. This is in contrast to a shift-reduce dependency parser in which a shift action typically just pushes a word onto the stack.

The COMBINE-X action pops the top two nodes off the stack, and combines them into a new node, which is pushed back on the stack. The category of the new node is X. A COMBINE action corresponds to a combinatory rule in the CCG grammar (or one of the additional punctuation or type-changing rules), which is applied to the categories of the top two nodes on the stack.

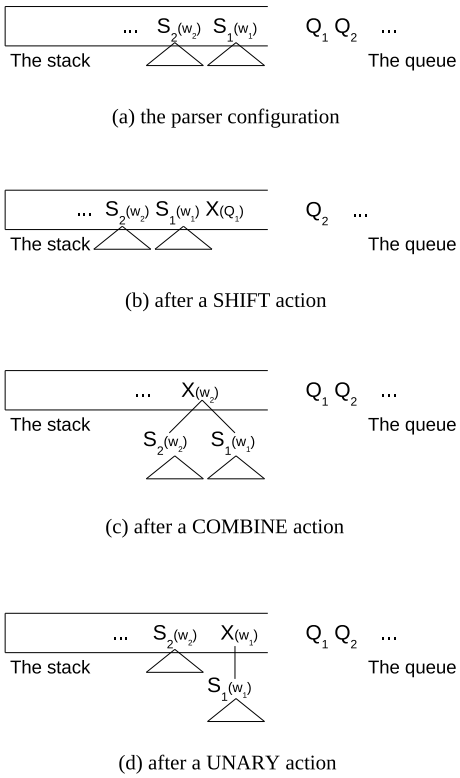

Figure 1: The parser configuration and set of actions.

The UNARY-X action pops the top of the stack, transforms it into a new node with category X, and pushes the new node onto the stack. A UNARY action corresponds to a unary type-changing or type-raising rule in the CCG grammar, which is applied to the category on top of the stack.

The FINISH action terminates the parsing process; it can be applied when all input words have been shifted onto the stack. Note that the FINISH action can be applied when the stack contains more than one node, in which case the parser produces a set of partial derivation trees, each corresponding to a node on the stack. This sometimes happens when a full derivation tree cannot be built due to supertagging errors, and provides a graceful solution to the problem of producing high-quality fragmentary parses when necessary.

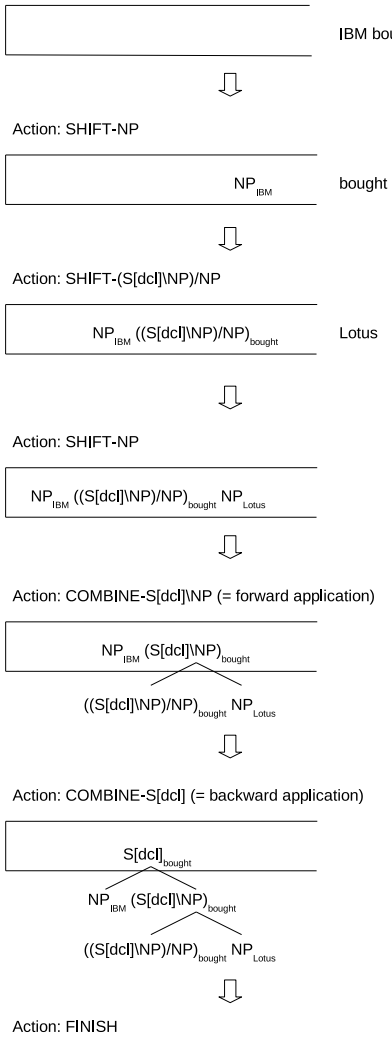

Figure 2: An example parsing process.

Figure 2 shows the shift-reduce parsing process for the example sentence "IBM bought Lotus". First the word "IBM" is shifted onto the stack as an NP; then "bought" is shifted as a transitive verb looking for its object NP on the right and subject NP on the left ((S[dcl]\NP)/NP); and then "Lotus" is shifted as an NP. Then "bought" is combined with its object "Lotus" resulting in a verb phrase looking for its subject on the left (S[dcl]\NP). Finally, the resulting verb phrase is combined with its subject, resulting in a declarative sentence (S[dcl]).

A key difference with previous work on shift-reduce dependency (Nivre et al., 2006) and CFG (Sagae and Lavie, 2006b) parsing is that, for CCG, there are many more shift actions – a shift action for each word-lexical category pair. Given the amount of syntactic information in the lexical categories, the choice of correct category, from those supplied by the supertagger, is often a difficult one, and often a choice best left to the parsing model. The C&C parser solves this problem by building the complete packed chart consistent with the lexical categories supplied by the supertagger, leaving the selection of the lexical categories to the Viterbi algorithm. For the shift-reduce parser the choice is also left to the parsing model, but in contrast to C&C the correct lexical category could be lost at any point in the heuristic search process. Hence it is perhaps surprising that we are able to achieve a high parsing accuracy of 85.5%, given a relatively small beam size.

4 Decoding

Greedy local search (Yamada and Matsumoto, 2003; Sagae and Lavie, 2005; Nivre and Scholz, 2004) has typically been used for decoding in shift-reduce parsers, while beam-search has recently been applied as an alternative to reduce error-propagation (Johansson and Nugues, 2007; Zhang and Clark, 2008; Zhang and Clark, 2009; Huang et al., 2009). Both greedy local search and beam-search have linear time complexity. We use beam-search in our CCG parser.

To formulate the decoding algorithm, we define a *candidate item* as a tuple $\langle S, Q, F \rangle$, where S represents the stack with partial derivations that have been built, Q represents the queue of incoming words that have not been processed, and F is a boolean value that represents whether the candidate item has been finished. A candidate item is *finished* if and only if the `FINISH` action has been applied to it, and no more actions can be applied to a candidate item after it reaches the finished status. Given an input sentence, we define the *start item* as the unfinished item with an empty stack and the whole input sentence as the incoming words. A derivation is built from the start item by repeated applications of actions until the item is finished.

To apply beam-search, an agenda is used to hold the N-best partial (unfinished) candidate items at each parsing step. A separate *candidate output* is

```
function DECODE(input, agenda, list, N,
                grammar, candidate_output):
  agenda.clear()
  agenda.insert(GETSTARTITEM(input))
  candidate_output = NONE
  while not agenda.empty():
    list.clear()
    for item in agenda:
      for action in grammar.getActions(item):
        item' = item.apply(action)
        if item'.F == TRUE:
          if candidate_output == NONE or
            item'.score > candidate_output.score:
            candidate_output = item'
          else:
            list.append(item')
    agenda.clear()
    agenda.insert(list.best(N))
```

Figure 3: The decoding algorithm; N is the agenda size

	feature templates
1	S_0wp, S_0c, S_0pc, S_0wc, S_1wp, S_1c, S_1pc, S_1wc, S_2pc, S_2wc, S_3pc, S_3wc,
2	Q_0wp, Q_1wp, Q_2wp, Q_3wp,
3	S_0Lpc, S_0Lwc, S_0Rpc, S_0Rwc, S_0Upc, S_0Uwc, S_1Lpc, S_1Lwc, S_1Rpc, S_1Rwc, S_1Upc, S_1Uwc,
4	S_0wcS_1wc, S_0cS_1w, S_0wS_1c, S_0cS_1c, S_0wcQ_0wp, S_0cQ_0wp, S_0wcQ_0p, S_0cQ_0p, S_1wcQ_0wp, S_1cQ_0wp, S_1wcQ_0p, S_1cQ_0p,
5	S_0wcS_1cQ_0p, S_0cS_1wcQ_0p, S_0cS_1cQ_0wp, S_0cS_1cQ_0p, S_0pS_1pQ_0p, S_0wcQ_0pQ_1p, S_0cQ_0wpQ_1p, S_0cQ_0pQ_1wp, S_0cQ_0pQ_1p, S_0pQ_0pQ_1p, S_0wcS_1cS_2c, S_0cS_1wcS_2c, S_0cS_1cS_2wc, S_0cS_1cS_2c, S_0pS_1pS_2p,
6	S_0cS_0HcS_0Lc, S_0cS_0HcS_0Rc, S_1cS_1HcS_1Rc, S_0cS_0RcQ_0p, S_0cS_0RcQ_0w, S_0cS_0LcS_1c, S_0cS_0LcS_1w, S_0cS_1cS_1Rc, S_0wS_1cS_1Rc.

Table 1: Feature templates.

used to record the current best finished item that has been found, since candidate items can be finished at different steps. Initially the agenda contains only the start item, and the *candidate output* is set to none. At each step during parsing, each candidate item from the agenda is extended in all possible ways by applying one action according to the grammar, and a number of new candidate items are generated. If a newly generated candidate is finished, it is compared with the current *candidate output*. If the candidate output is none or the score of the newly generated candidate is higher than the score of the candidate output, the candidate output is replaced with the newly generated item; otherwise the newly generated item is discarded. If the newly generated candidate is unfinished, it is appended to a *list* of newly generated partial candidates. After all candidate items from the agenda have been processed, the agenda is cleared and the N-best items from the list are put on the agenda. Then the list is cleared and the parser moves on to the next step. This process repeats until the agenda is empty (which means that no new items have been generated in the previous step), and the candidate output is the final derivation. Pseudocode for the algorithm is shown in Figure 3.

5 Model and Training

We use a global linear model to score candidate items, trained discriminatively with the averaged perceptron (Collins, 2002). Features for a (finished or partial) candidate are extracted from each action that have been applied to build the candidate. Following Collins and Roark (2004), we apply the "early update" strategy to perceptron training: at any step during decoding, if neither the candidate output nor any item in the agenda is correct, decoding is stopped and the parameters are updated using the current highest scored item in the agenda or the candidate output, whichever has the higher score.

Table 1 shows the feature templates used by the parser. The symbols S_0, S_1, S_2 and S_3 in the table represent the top four nodes on the stack (if existent), and Q_0, Q_1, Q_2 and Q_3 represent the front four words in the incoming queue (if existent). S_0H and S_1H represent the subnodes of S_0 and S_1 that have the lexical head of S_0 and S_1, respectively. S_0L represents the left subnode of S_0, when the lexical head is from the right subnode. S_0R and S_1R represent the right subnode of S_0 and S_1, respectively,

when the lexical head is from the left subnode. If S_0 is built by a UNARY action, S_0U represents the only subnode of S_0. The symbols w, p and c represent the word, the POS, and the CCG category, respectively.

These rich feature templates produce a large number of features: 36 million after the first training iteration, compared to around 0.5 million in the C&C parser.

6 Experiments

Our experiments were performed using CCGBank (Hockenmaier and Steedman, 2007), which was split into three subsets for training (Sections 02–21), development testing (Section 00) and the final test (Section 23). Extracted from the training data, the CCG grammar used by our parser consists of 3070 binary rule instances and 191 unary rule instances.

We compute F-scores over labeled CCG dependencies and also lexical category accuracy. CCG dependencies are defined in terms of lexical categories, by numbering each argument slot in a complex category. For example, the first NP in a transitive verb category is a CCG dependency relation, corresponding to the subject of the verb. Clark and Curran (2007) gives a more precise definition. We use the `generate` script from the C&C tools[3] to transform derivations into CCG dependencies.

There is a mismatch between the grammar that `generate` uses, which is the same grammar as the C&C parser, and the grammar we extract from CCGbank, which contains more rule instances. Hence `generate` is unable to produce dependencies for some of the derivations our shift-reduce parser produces. In order to allow `generate` to process all derivations from the shift-reduce parser, we repeatedly removed rules that the `generate` script cannot handle from our grammar, until all derivations in the development data could be dealt with. In fact, this procedure potentially reduces the accuracy of the shift-reduce parser, but the effect is comparatively small because only about 4% of the development and test sentences contain rules that are not handled by the `generate` script.

All experiments were performed using automatically assigned POS-tags, with 10-fold cross validation used to assign POS-tags and lexical categories to the training data. At the supertagging stage, multiple lexical categories are assigned to each word in the input. For each word, the supertagger assigns all lexical categories whose forward-backward probability is above $\beta \cdot max$, where max is the highest lexical category probability for the word, and β is a threshold parameter. To give the parser a reasonable freedom in lexical category disambiguation, we used a small β value of 0.0001, which results in 3.6 lexical categories being assigned to each word on average in the training data. For training, but not testing, we also added the correct lexical category to the list of lexical categories for a word in cases when it was not provided by the supertagger.

Increasing the size of the beam in the parser beam search leads to higher accuracies but slower running time. In our development experiments, the accuracy improvement became small when the beam size reached 16, and so we set the size of the beam to 16 for the remainder of the experiments.

6.1 Development test accuracies

Table 2 shows the labeled precision (lp), recall (lr), F-score (lf), sentence-level accuracy (lsent) and lexical category accuracy (cats) of our parser and the C&C parser on the development data. We ran the C&C parser using the normal-form model (we reproduced the numbers reported in Clark and Curran (2007)), and copied the results of the hybrid model from Clark and Curran (2007), since the hybrid model is not part of the public release.

The accuracy of our parser is much better when evaluated on all sentences, partly because C&C failed on 0.94% of the data due to the failure to produce a spanning analysis. Our shift-reduce parser does not suffer from this problem because it produces fragmentary analyses for those cases. When evaluated on only those sentences that C&C could analyze, our parser gave 0.29% higher F-score. Our shift-reduce parser also gave higher accuracies on lexical category assignment. The sentence accuracy of our shift-reduce parser is also higher than C&C, which confirms that our shift-reduce parser produces reasonable sentence-level analyses, despite the possibility for fragmentary analysis.

[3] Available at http://svn.ask.it.usyd.edu.au/trac/candc/wiki; we used the `generate` and `evaluate` scripts, as well as the C&C parser, for evaluation and comparison.

	lp.	lr.	lf.	lsent.	cats.	evaluated on
shift-reduce	87.15%	82.95%	85.00%	33.82%	92.77%	all sentences
C&C (normal-form)	85.22%	82.52%	83.85%	31.63%	92.40%	all sentences
shift-reduce	87.55%	83.63%	85.54%	34.14%	93.11%	99.06% (C&C coverage)
C&C (hybrid)	–	–	85.25%	–	–	99.06% (C&C coverage)
C&C (normal-form)	85.22%	84.29%	84.76%	31.93%	92.83%	99.06% (C&C coverage)

Table 2: Accuracies on the development test data.

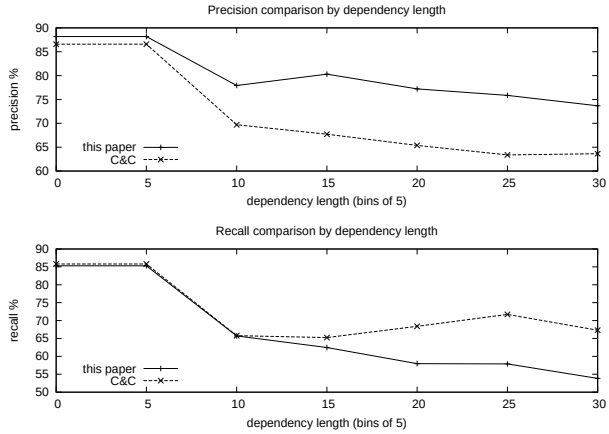

Figure 4: P & R scores relative to dependency length.

6.2 Error comparison with C&C parser

Our shift-reduce parser and the chart-based C&C parser offer two different solutions to the CCG parsing problem. The comparison reported in this section is similar to the comparison between the chart-based MSTParser (McDonald et al., 2005) and shift-reduce MaltParser (Nivre et al., 2006) for dependency parsing. We follow McDonald and Nivre (2007) and characterize the errors of the two parsers by sentence and dependency length and dependency type.

We measured precision, recall and F-score relative to different sentence lengths. Both parsers performed better on shorter sentences, as expected. Our shift-reduce parser performed consistently better than C&C on all sentence lengths, and there was no significant difference in the rate of performance degradation between the parsers as the sentence length increased.

Figure 4 shows the comparison of labeled precision and recall relative to the dependency length (i.e. the number of words between the head and dependent), in bins of size 5 (e.g. the point at x=5 shows the precision or recall for dependency lengths 1 – 5). This experiment was performed using the normal-form version of the C&C parser, and the evaluation was on the sentences for which C&C gave an analysis. The number of dependencies drops when the dependency length increases; there are 141, 180 and 124 dependencies from the gold-standard, C&C output and our shift-reduce parser output, respectively, when the dependency length is between 21 and 25, inclusive. The numbers drop to 47, 56 and 36 when the dependency length is between 26 and 30. The recall of our parser drops more quickly as the dependency length grows beyond 15. A likely reason is that the recovery of longer-range dependencies requires more processing steps, increasing the chance of the correct structure being thrown off the beam. In contrast, the precision did not drop more quickly than C&C, and in fact is consistently higher than C&C across all dependency lengths, which reflects the fact that the long range dependencies our parser managed to recover are comparatively reliable.

Table 3 shows the comparison of labeled precision (lp), recall (lr) and F-score (lf) for the most common CCG dependency types. The numbers for C&C are for the hybrid model, copied from Clark and Curran (2007). While our shift-reduce parser gave higher precision for almost all categories, it gave higher recall on only half of them, but higher F-scores for all but one dependency type.

6.3 Final results

Table 4 shows the accuracies on the test data. The numbers for the normal-form model are evaluated by running the publicly available parser, while those for the hybrid dependency model are from Clark and Curran (2007). Evaluated on all sentences, the accuracies of our parser are much higher than the C&C parser, since the C&C parser failed to produce any output for 10 sentences. When evaluating both

category	arg	lp. (o)	lp. (C)	lr. (o)	lr. (C)	lf. (o)	lf. (C)	freq.
N/N	1	**95.77%**	95.28%	**95.79%**	95.62%	**95.78%**	95.45%	7288
NP/N	1	**96.70%**	96.57%	**96.59%**	96.03%	**96.65%**	96.30%	4101
(NP\NP)/NP	2	**83.19%**	82.17%	**89.24%**	88.90%	**86.11%**	85.40%	2379
(NP\NP)/NP	1	**82.53%**	81.58%	**87.99%**	85.74%	**85.17%**	83.61%	2174
((S\NP)\(S\NP))/NP	3	**77.60%**	71.94%	71.58%	**73.32%**	**74.47%**	72.63%	1147
((S\NP)\(S\NP))/NP	2	**76.30%**	70.92%	70.60%	**71.93%**	**73.34%**	71.42%	1058
((S[dcl]\NP)/NP	2	**85.60%**	81.57%	84.30%	**86.37%**	**84.95%**	83.90%	917
PP/NP	1	73.76%	**75.06%**	**72.83%**	70.09%	**73.29%**	72.49%	876
((S[dcl]\NP)/NP	1	**85.32%**	81.62%	82.00%	**85.55%**	**83.63%**	83.54%	872
((S\NP)\(S\NP))	2	84.44%	**86.85%**	86.60%	**86.73%**	85.51%	**86.79%**	746

Table 3: Accuracy comparison on the most common CCG dependency types. (o) – our parser; (C) – C&C (hybrid)

	lp.	lr.	lf.	lsent.	cats.	evaluated
shift-reduce	87.43%	83.61%	85.48%	35.19%	93.12%	all sentences
C&C (normal-form)	85.58%	82.85%	84.20%	32.90%	92.84%	all sentences
shift-reduce	87.43%	83.71%	85.53%	35.34%	93.15%	99.58% (C&C coverage)
C&C (hybrid)	86.17%	84.74%	85.45%	32.92%	92.98%	99.58% (C&C coverage)
C&C (normal-form)	85.48%	84.60%	85.04%	33.08%	92.86%	99.58% (C&C coverage)
F&P (Petrov I-5)*	86.29%	85.73%	86.01%	–	–	– (F&P ∩ C&C coverage; 96.65% on dev. test)
C&C hybrid*	86.46%	85.11%	85.78%	–	–	– (F&P ∩ C&C coverage; 96.65% on dev. test)

Table 4: Comparison with C&C; final test. * – not directly comparable.

parsers on the sentences for which C&C produces an analysis, our parser still gave the highest accuracies. The shift-reduce parser gave higher precision, and lower recall, than C&C; it also gave higher sentence-level and lexical category accuracy.

The last two rows in the table show the accuracies of Fowler and Penn (2010) (F&P), who applied the CFG parser of Petrov and Klein (2007) to CCG, and the corresponding accuracies for the C&C parser on the same test sentences. F&P can be treated as another chart-based parser; their evaluation is based on the sentences for which both their parser and C&C produced dependencies (or more specifically those sentences for which `generate` could produce dependencies), and is not directly comparable with ours, especially considering that their test set is smaller and potentially slightly easier.

The final comparison is parser speed. The shift-reduce parser is linear-time (in both sentence length and beam size), and can analyse over 10 sentences per second on a 2GHz CPU, with a beam of 16, which compares very well with other constituency parsers. However, this is no faster than the chart-based C&C parser, although speed comparisons are difficult because of implementation differences (C&C uses heavily engineered C++ with a focus on efficiency).

7 Related Work

Sagae and Lavie (2006a) describes a shift-reduce parser for the Penn Treebank parsing task which uses best-first search to allow some ambiguity into the parsing process. Differences with our approach are that we use a beam, rather than best-first, search; we use a global model rather than local models chained together; and finally, our results surpass the best published results on the CCG parsing task, whereas Sagae and Lavie (2006a) matched the best PTB results only by using a parser combination.

Matsuzaki et al. (2007) describes similar work to ours but using an automatically-extracted HPSG, rather than CCG, grammar. They also use the generalised perceptron to train a disambiguation model. One difference is that Matsuzaki et al. (2007) use an approximating CFG, in addition to the supertagger, to improve the efficiency of the parser.

Ninomiya et al. (2009) (and Ninomiya et al. (2010)) describe a greedy shift-reduce parser for HPSG, in which a single action is chosen at each parsing step, allowing the possibility of highly efficient parsing. Since the HPSG grammar has relatively tight constraints, similar to CCG, the possibility arises that a spanning analysis cannot be found for some sentences. Our approach to this problem was to allow the parser to return a fragmentary analysis; Ninomiya et al. (2009) adopt a different approach based on default unification.

Finally, our work is similar to the comparison of the chart-based MSTParser (McDonald et al., 2005) and shift-reduce MaltParser (Nivre et al., 2006) for dependency parsing. MSTParser can perform exhaustive search, given certain feature restrictions, because the complexity of the parsing task is lower than for constituent parsing. C&C can perform exhaustive search because the supertagger has already reduced the search space. We also found that approximate heuristic search for shift-reduce parsing, utilising a rich feature space, can match the performance of the optimal chart-based parser, as well as similar error profiles for the two CCG parsers compared to the two dependency parsers.

8 Conclusion

This is the first work to present competitive results for CCG using a transition-based parser, filling a gap in the CCG parsing literature. Considered in terms of the wider parsing problem, we have shown that state-of-the-art parsing results can be obtained using a global discriminative model, one of the few papers to do so without using a generative baseline as a feature. The comparison with C&C also allowed us to compare a shift-reduce parser based on heuristic beam search utilising a rich feature set with an optimal chart-based parser whose features are restricted by dynamic programming, with favourable results for the shift-reduce parser.

The complementary errors made by the chart-based and shift-reduce parsers opens the possibility of effective parser combination, following similar work for dependency parsing.

The parser code can be downloaded at `http://www.sourceforge.net/projects/zpar`, version 0.5.

Acknowledgements

We thank the anonymous reviewers for their suggestions. Yue Zhang and Stephen Clark are supported by the European Union Seventh Framework Programme (FP7-ICT-2009-4) under grant agreement no. 247762.

References

A. E. Ades and M. Steedman. 1982. On the order of words. *Linguistics and Philosophy*, pages 517 – 558.

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

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 and James R. Curran. 2009. Comparing the accuracy of CCG and Penn Treebank parsers. In *Proceedings of ACL-2009 (short papers)*, pages 53–56, Singapore.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*, pages 111–118, Barcelona, Spain.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, USA.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Feature-based, conditional random field parsing. In *Proceedings of the 46th Meeting of the ACL*, pages 959–967, Columbus, Ohio.

Timothy A. D. Fowler and Gerald Penn. 2010. Accurate context-free parsing with Combinatory Categorial Grammar. In *Proceedings of ACL-2010*, Uppsala, Sweden.

H. Hassan, K. Sima'an, and A. Way. 2008. A syntactic language model based on incremental CCG parsing. In *Proceedings of the Second IEEE Spoken Language Technology Workshop*, Goa, India.

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.

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

Liang Huang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce

parsing. In *Proceedings of the 2009 EMNLP Conference*, pages 1222–1231, Singapore.

Richard Johansson and Pierre Nugues. 2007. Incremental dependency parsing using online learning. In *Proceedings of the CoNLL/EMNLP Conference*, pages 1134–1138, Prague, Czech Republic.

Takuya Matsuzaki, Yusuke Miyao, and Jun ichi Tsujii. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of IJCAI-07*, pages 1671–1676, Hyderabad, India.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP/CoNLL*, pages 122–131, Prague, Czech Republic.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Meeting of the ACL*, pages 91–98, Michigan, Ann Arbor.

Yusuke Miyao and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd meeting of the ACL*, pages 83–90, University of Michigan, Ann Arbor.

Takashi Ninomiya, Takuya Matsuzaki, Nobuyuki Shimizu, and Hiroshi Nakagawa. 2009. Deterministic shift-reduce parsing for unification-based grammars by using default unification. In *Proceedings of EACL-09*, pages 603–611, Athens, Greece.

Takashi Ninomiya, Takuya Matsuzaki, Nobuyuki Shimizu, and Hiroshi Nakagawa. 2010. Deterministic shift-reduce parsing for unification-based grammars. *Journal of Natural Language Engineering*, DOI:10.1017/S1351324910000240.

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

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of CoNLL*, pages 221–225, New York, USA.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of HLT/NAACL*, pages 404–411, Rochester, New York, April.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the ACL*, pages 271–278, Philadelphia, PA.

Laura Rimell, Stephen Clark, and Mark Steedman. 2009. Unbounded dependency recovery for parser evaluation. In *Proceedings of EMNLP-09*, pages 813–821, Singapore.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of IWPT*, pages 125–132, Vancouver, Canada.

Kenji Sagae and Alon Lavie. 2006a. A best-first probabilistic shift-reduce parser. In *Proceedings of COLING/ACL poster session*, pages 691–698, Sydney, Australia, July.

Kenji Sagae and Alon Lavie. 2006b. Parser combination by reparsing. In *Proceedings of HLT/NAACL, Companion Volume: Short Papers*, pages 129–132, New York, USA.

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

David Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.

Michael White and Rajakrishnan Rajkumar. 2009. Perceptron reranking for CCG realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 410–419, Singapore.

H Yamada and Y Matsumoto. 2003. Statistical dependency analysis using support vector machines. In *Proceedings of IWPT*, Nancy, France.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP-08*, Hawaii, USA.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the Chinese Treebank using a global discriminative model. In *Proceedings of IWPT*, Paris, France, October.

Yi Zhang, Valia Kordoni, and Erin Fitzgerald. 2007. Partial parse selection for robust deep processing. In *Proceedings of the ACL 2007 Workshop on Deep Linguistic Processing*, Prague, Czech Republic.