Building Deep Dependency Structures with a Wide-Coverage CCG Parser

Stephen Clark, Julia Hockenmaier and **Mark Steedman**
Division of Informatics
University of Edinburgh
Edinburgh EH8 9LW, UK
{stephenc, julia, steedman}@cogsci.ed.ac.uk

Abstract

This paper describes a wide-coverage statistical parser that uses Combinatory Categorial Grammar (CCG) to derive dependency structures. The parser differs from most existing wide-coverage treebank parsers in capturing the long-range dependencies inherent in constructions such as coordination, extraction, raising and control, as well as the standard local predicate-argument dependencies. A set of dependency structures used for training and testing the parser is obtained from a treebank of CCG normal-form derivations, which have been derived (semi-) automatically from the Penn Treebank. The parser correctly recovers over 80% of labelled dependencies, and around 90% of unlabelled dependencies.

1 Introduction

Most recent wide-coverage statistical parsers have used models based on lexical dependencies (e.g. Collins (1999), Charniak (2000)). However, the dependencies are typically derived from a context-free phrase structure tree using simple head percolation heuristics. This approach does not work well for the long-range dependencies involved in raising, control, extraction and coordination, all of which are common in text such as the Wall Street Journal.

Chiang (2000) uses Tree Adjoining Grammar as an alternative to context-free grammar, and here we use another "mildly context-sensitive" formalism, Combinatory Categorial Grammar (CCG, Steedman (2000)), which arguably provides the most linguistically satisfactory account of the dependencies inherent in coordinate constructions and extraction phenomena. The potential advantage from using such an expressive grammar is to facilitate recovery of such *unbounded* dependencies. As well as having a potential impact on the accuracy of the parser, recovering such dependencies may make the output more useful.

CCG is unlike other formalisms in that the standard predicate-argument relations relevant to interpretation can be derived via extremely non-standard surface derivations. This impacts on how best to define a probability model for CCG, since the "spurious ambiguity" of CCG derivations may lead to an exponential number of derivations for a given constituent. In addition, some of the spurious derivations may not be present in the training data. One solution is to consider only the normal-form (Eisner, 1996a) derivation, which is the route taken in Hockenmaier and Steedman (2002b).[1]

Another problem with the non-standard surface derivations is that the standard PARSEVAL performance measures over such derivations are uninformative (Clark and Hockenmaier, 2002). Such measures have been criticised by Lin (1995) and Carroll et al. (1998), who propose recovery of head-dependencies characterising predicate-argument relations as a more meaningful measure.

If the end-result of parsing is interpretable predicate-argument structure or the related dependency structure, then the question arises: *why build derivation structure at all?* A CCG parser can directly build derived structures, including long-

[1] Another, more speculative, possibility is to treat the alternative derivations as hidden and apply the EM algorithm.

range dependencies. These derived structures can be of any form we like—for example, they could in principle be standard Penn Treebank structures. Since we are interested in dependency-based parser evaluation, our parser currently builds dependency structures. Furthermore, since we want to model the dependencies in such structures, the probability model is defined over these structures rather than the derivation.

The training and testing material for this CCG parser is a treebank of dependency structures, which have been derived from a set of CCG derivations developed for use with another (normal-form) CCG parser (Hockenmaier and Steedman, 2002b). The treebank of derivations, which we call CCGbank (Hockenmaier and Steedman, 2002a), was in turn derived (semi-)automatically from the hand-annotated Penn Treebank.

2 The Grammar

In CCG, most language-specific aspects of the grammar are specified in the lexicon, in the form of syntactic categories that identify a lexical item as either a *functor* or *argument*. For the functors, the category specifies the type and directionality of the arguments and the type of the result. For example, the following category for the transitive verb *bought* specifies its first argument as a noun phrase (*NP*) to its right and its second argument as an *NP* to its left, and its result as a sentence:

(1) bought := $(S\backslash NP)/NP$

For parsing purposes, we extend CCG categories to express category features, and head-word and dependency information directly, as follows:

(2) bought := $(S[dcl]_{bought}\backslash NP_{1})/NP_{2}$

The feature $[dcl]$ specifies the category's S result as a declarative sentence, *bought* identifies its head, and the numbers denote dependency relations. Heads and dependencies are always marked up on atomic categories (S, N, NP, PP, and $conj$ in our implementation).

The categories are combined using a small set of typed combinatory rules, such as functional application and composition (see Steedman (2000) for details). Derivations are written as follows, with underlines indicating combinatory reduction and arrows indicating the direction of the application:

(3)
$$\begin{array}{ccc}
\text{Marks} & \text{bought} & \text{Brooks} \\
\hline NP_{Marks} & (S[dcl]_{bought}\backslash NP_1)/NP_2 & NP_{Brooks} \\
 & \multicolumn{2}{c}{\overline{S[dcl]_{bought}\backslash NP_1}^{\,>}} \\
\multicolumn{3}{c}{\overline{S[dcl]_{bought}}^{\,<}}
\end{array}$$

Formally, a dependency is defined as a 4-tuple: $\langle h_f, f, s, h_a\rangle$, where h_f is the head word of the functor,[2] f is the functor category (extended with head and dependency information), s is the argument slot, and h_a is the head word of the argument—for example, the following is the object dependency yielded by the first step of derivation (3):

(4) $\langle \text{bought}, (S[dcl]_{bought}\backslash NP_{1})/NP_{2}, 2, \text{Brooks}\rangle$

Variables can also be used to denote heads, and used via unification to pass head information from one category to another. For example, the expanded category for the control verb *persuade* is as follows:

(5) persuade := $((S[dcl]_{persuade}\backslash NP_{1})/(S[to]_{2}\backslash NP_{X}))/NP_{X,3}$

The head of the infinitival complement's subject is identified with the head of the object, using the variable *X*. Unification then "passes" the head of the object to the subject of the infinitival, as in standard unification-based accounts of control.[3]

The kinds of lexical items that use the head passing mechanism are raising, auxiliary and control verbs, modifiers, and relative pronouns. Among the constructions that project unbounded dependencies are relativisation and right node raising. The following category for the relative pronoun category (for words such as *who, which, that*) shows how heads are co-indexed for object-extraction:

(6) who := $(NP_{X}\backslash NP_{X,1})/(S[dcl]_{2}/NP_{X})$

The derivation for the phrase *The company that Marks wants to buy* is given in Figure 1 (with the features on *S* categories removed to save space, and the constant heads reduced to the first letter). Type-raising (**T**) and functional composition (**B**), along

[2] Note that the functor does not always correspond to the linguistic notion of a head.

[3] The extension of CCG categories in the lexicon and the labelled data is simplified in the current system to make it entirely automatic. For example, any word with the same category (5) as *persuade* gets the object-control extension. In certain rare cases (such as *promise*) this gives semantically incorrect dependencies in both the grammar and the data (*promise Brooks to go* has a structure meaning *promise Brooks that Brooks will go*).

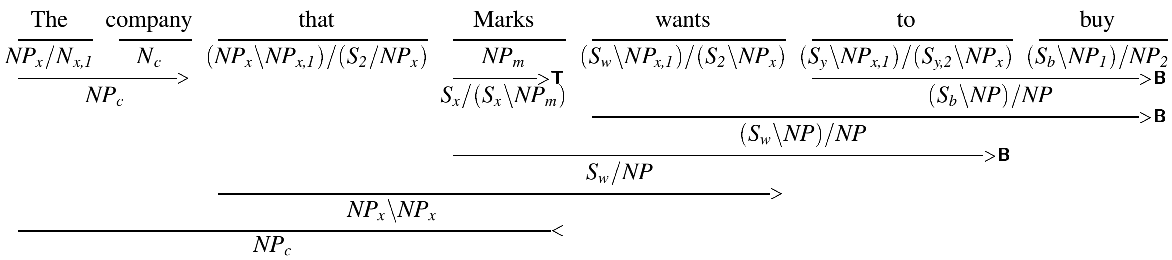
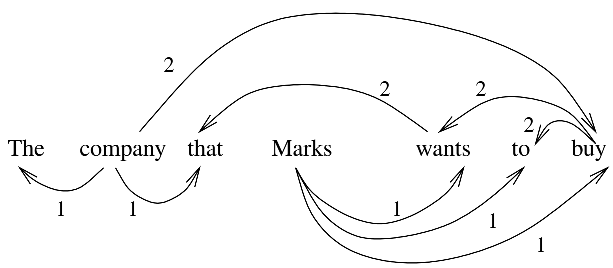


Figure 1: Relative clause derivation

with co-indexing of heads, mediate transmission of the head of the *NP the company* onto the object of *buy*. The corresponding dependencies are given in the following figure, with the convention that arcs point away from arguments. The relevant argument slot in the functor category labels the arcs.



Note that we encode the subject argument of the *to* category as a dependency relation (*Marks* is a "subject" of *to*), since our philosophy at this stage is to encode every argument as a dependency, where possible. The number of dependency types may be reduced in future work.

3 The Probability Model

The DAG-like nature of the dependency structures makes it difficult to apply generative modelling techniques (Abney, 1997; Johnson et al., 1999), so we have defined a conditional model, similar to the model of Collins (1996) (see also the conditional model in Eisner (1996b)). While the model of Collins (1996) is technically unsound (Collins, 1999), our aim at this stage is to demonstrate that accurate, efficient wide-coverage parsing is possible with CCG, even with an over-simplified statistical model. Future work will look at alternative models.[4]

[4]The reentrancies creating the DAG-like structures are fairly limited, and moreover determined by the lexical categories. We conjecture that it is possible to define a generative model that includes the deep dependencies.

The parse selection component must choose the most probable dependency structure, given the sentence S. A sentence $S = \langle w_1, t_1 \rangle, \langle w_2, t_2 \rangle, \ldots \langle w_n, t_n \rangle$ is assumed to be a sequence of word, pos-tag pairs. For our purposes, a dependency structure π is a $\langle C, D \rangle$ pair, where $C = c_1, c_2 \ldots c_n$ is the sequence of categories assigned to the words, and $D = \left\{ \langle h_{f_i}, f_i, s_i, h_{a_i} \rangle \mid i = 1, \ldots m \right\}$ is the set of dependencies. The probability of a dependency structure can be written as follows:

(7) $P(\pi) = P(C, D|S) = P(C|S)P(D|C, S)$

The probability $P(C|S)$ can be approximated as follows:

(8) $P(C|S) \approx \prod_{i=1}^{n} P(c_i|X_i)$

where X_i is the local context for the ith word. We have explained elsewhere (Clark, 2002) how suitable features can be defined in terms of the $\langle$word, pos-tag$\rangle$ pairs in the context, and how maximum entropy techniques can be used to estimate the probabilities, following Ratnaparkhi (1996).

We assume that each argument slot in the category sequence is filled independently, and write $P(D|C, S)$ as follows:

(9) $P(D|C, S) = \prod_{i=1}^{m} P(h_{a_i}|C, S)$

where h_{a_i} is the head word filling the argument slot of the ith dependency, and m is the number of dependencies entailed by the category sequence C.

3.1 Estimating the dependency probabilities

The estimation method is based on Collins (1996). We assume that the probability of a dependency only depends on those words involved in the dependency, together with their categories. We follow Collins and base the estimate of a dependency probability on the following intuition: given a pair of words, with a pair of categories, which are in the same sentence, what is the probability that the words are in a particular dependency relationship?

We again follow Collins in defining the following functions, where $\mathcal{W}$ is the set of words in the data, and $\mathcal{C}$ is the set of lexical categories.

- $C(\langle a,b\rangle,\langle c,d\rangle)$ for $a,c \in \mathcal{W}$ and $b,d \in \mathcal{C}$ is the number of times that word-category pairs $\langle a,b\rangle$ and $\langle c,d\rangle$ are in the same word-category sequence in the training data.

- $C(R,\langle a,b\rangle,\langle c,d\rangle)$ is the number of times that $\langle a,b\rangle$ and $\langle c,d\rangle$ are in the same word-category sequence, with a and c in dependency relation R.

- $F(R|\langle a,b\rangle,\langle c,d\rangle)$ is the probability that a and c are in dependency relation R, given that $\langle a,b\rangle$ and $\langle c,d\rangle$ are in the same word-category sequence.

The relative frequency estimate of the probability $F(R|\langle a,b\rangle,\langle c,d\rangle)$ is as follows:

(10) $\hat{F}(R|\langle a,b\rangle,\langle c,d\rangle) = \frac{C(R,\langle a,b\rangle,\langle c,d\rangle)}{C(\langle a,b\rangle,\langle c,d\rangle)}$

The probability $P(h_{a_i}|C,S)$ can now be approximated as follows:

(11) $P(h_{a_i}|C,S) \approx \frac{\hat{F}(R|\langle h_{f_i},f_i\rangle,\langle h_{a_i},c_{a_i}\rangle)}{\sum_{j=1}^{n}\hat{F}(R|\langle h_{f_i},f_i\rangle,\langle w_j,c_j\rangle)}$

where c_{a_i} is the lexical category of the argument head a_i. The normalising factor ensures that the probabilities for each argument slot sum to one over all the word-category pairs in the sequence.[5] This factor is constant for the given category sequence, but not for different category sequences. However, the dependency structures with high enough $P(C|S)$ to be among the highest probability structures are likely to have similar category sequences. Thus we ignore the normalisation factor, thereby simplifying the parsing process. (A similar argument is used by Collins (1996) in the context of his parsing model.)

The estimate in equation 10 suffers from sparse data problems, and so a backing-off strategy is employed. We omit details here, but there are four levels of back-off: the first uses both words and both categories; the second uses only one of the words and both categories; the third uses the categories only; and a final level substitutes pos-tags for the categories.

One final point is that, in practice, the number of dependencies can vary for a given category sequence (because multiple arguments for the same slot can be introduced through coordination), and so a geometric mean of $p(\pi)$ is used as the ranking function, averaged by the number of dependencies in D.

4 The Parser

The parser analyses a sentence in two stages. First, in order to limit the number of categories assigned to each word in the sentence, a "supertagger" (Bangalore and Joshi, 1999) assigns to each word a small number of possible lexical categories. The supertagger (described in Clark (2002)) assigns to each word all categories whose probabilities are within some constant factor, β, of the highest probability category for that word, given the surrounding context. Note that the supertagger does not provide a single category sequence for each sentence, and the final sequence returned by the parser (along with the dependencies) is determined by the probability model described in the previous section. The supertagger is performing two roles: cutting down the search space explored by the parser, and providing the category-sequence model in equation 8.

The supertagger consults a "category dictionary" which contains, for each word, the set of categories the word was seen with in the data. If a word appears at least K times in the data, the supertagger only considers categories that appear in the word's category set, rather than all lexical categories.

The second parsing stage applies a CKY bottom-up chart-parsing algorithm, as described in Steedman (2000). The combinatory rules currently used by the parser are as follows: functional application (forward and backward), generalised forward composition, backward composition, generalised backward-crossed composition, and type-raising. There is also a coordination rule which conjoins categories of the same type.[6]

Type-raising is applied to the categories NP, PP, and $S[adj]\backslash NP$ (adjectival phrase); it is currently implemented by simply adding pre-defined sets of type-raised categories to the chart whenever an NP, PP or $S[adj]\backslash NP$ is present. The sets were chosen on the basis of the most frequent type-raising rule instantiations in sections 02-21 of the CCGbank, which resulted in 8 type-raised categories for NP,

[5] One of the problems with the model is that it is deficient, assigning probability mass to dependency structures not licensed by the grammar.

[6] Restrictions are placed on some of the rules, such as that given by Steedman (2000) for backward-crossed composition (p.62).

and 2 categories each for PP and $S[adj]\backslash NP$.

As well as combinatory rules, the parser also uses a number of lexical rules and rules involving punctuation. The set of rules consists of those occurring roughly more than 200 times in sections 02-21 of the CCGbank. For example, one rule used by the parser is the following:

(12) $S[ing]\backslash NP \Rightarrow NP_X\backslash NP_X$

This rule creates a nominal modifier from an *ing*-form of a verb phrase.

A set of rules allows the parser to deal with commas (all other punctuation is removed after the supertagging phase). For example, one kind of rule treats a comma as a conjunct, which allows the *NP* object in *John likes apples, bananas and pears* to have three heads, which can all be direct objects of *like*.[7]

The search space explored by the parser is reduced by exploiting the statistical model. First, a constituent is only placed in a chart cell if there is not already a constituent with the same head word, same category, and some dependency structure with a higher or equal score (where score is the geometric mean of the probability of the dependency structure). This tactic also has the effect of eliminating "spuriously ambiguous" entries from the chart—cf. Komagata (1997). Second, a constituent is only placed in a cell if the score for its dependency structure is within some factor, α, of the highest scoring dependency structure for that cell.

5 Experiments

Sections 02-21 of the CCGbank were used for training (39,161 sentences); section 00 for development (1,901 sentences); and section 23 for testing (2,379 sentences).[8] Sections 02-21 were also used to obtain the category set, by including all categories that appear at least 10 times, which resulted in a set of 398 category types.

The word-category sequences needed for estimating the probabilities in equation 8 can be read directly from the CCGbank. To obtain dependencies for estimating $P(D|C,S)$, we ran the parser over the trees, tracing out the combinatory rules applied during the derivation, and outputting the dependencies. This method was also applied to the trees in section 23 to provide the gold standard test set.

Not all trees produced dependency structures, since not all categories and type-changing rules in the CCGbank are encoded in the parser. We obtained dependency structures for roughly 95% of the trees in the data. For evaluation purposes, we increased the coverage on section 23 to 99.0% (2,352 sentences) by identifying the cause of the parse failures and adding the additional rules and categories when creating the gold-standard; so the final test set consisted of gold-standard dependency structures from 2,352 sentences. The coverage was increased to ensure the test set was representative of the full section. We emphasise that these additional rules and categories were not made available to the parser during testing, or used for training.

Initially the parser was run with $\beta = 0.01$ for the supertagger (an average of 3.8 categories per word), $K = 20$ for the category dictionary, and $\alpha = 0.001$ for the parser. A time-out was applied so that the parser was stopped if any sentence took longer than 2 CPU minutes to parse. With these parameters, 2,098 of the 2,352 sentences received some analysis, with 206 timing out and 48 failing to parse.

To deal with the 48 no-analysis cases, the cut-off for the category-dictionary, K, was increased to 100. Of the 48 cases, 23 sentences then received an analysis. To deal with the 206 time-out cases, β was increased to 0.05, which resulted in 181 of the 206 sentences then receiving an analysis, with 18 failing to parse, and 7 timing out. So overall, almost 98% of the 2,352 unseen sentences were given some analysis.

To return a single dependency structure, we chose the most probable structure from the $S[dcl]$ categories spanning the whole sentence. If there was no such category, all categories spanning the whole string were considered.

6 Results

To measure the performance of the parser, we compared the dependencies output by the parser with those in the gold standard, and computed precision

[7] These rules are currently applied deterministically. In future work we will investigate approaches which integrate the rule applications with the statistical model.

[8] A small number of sentences in the Penn Treebank do not appear in the CCGbank (see Hockenmaier and Steedman (2002a)).

and recall figures over the dependencies. Recall that a dependency is defined as a 4-tuple: a head of a functor, a functor category, an argument slot, and a head of an argument. Figures were calculated for labelled dependencies (LP,LR) and unlabelled dependencies (UP,UR). To obtain a point for a labelled dependency, each element of the 4-tuple must match exactly. Note that the category set we are using distinguishes around 400 distinct types; for example, tensed transitive *buy* is treated as a distinct category from infinitival transitive *buy*. Thus this evaluation criterion is much more stringent than that for a standard pos-tag label-set (there are around 50 pos-tags used in the Penn Treebank).

To obtain a point for an unlabelled dependency, the heads of the functor and argument must appear together in some relation (either as functor or argument) for the relevant sentence in the gold standard. The results are shown in Table 1, with an additional column giving the category accuracy.

	LP %	LR %	UP %	UR%	category %
no Δ	81.3	82.1	89.1	90.1	90.6
with Δ	81.9	81.8	90.1	89.9	90.3

Table 1: Overall dependency results for section 23

As an additional experiment, we conditioned the dependency probabilities in 10 on a "distance measure" (Δ). Distance has been shown to be a useful feature for context-free treebank style parsers (e.g. Collins (1996), Collins (1999)), although our hypothesis was that it would be less useful here, because the CCG grammar provides many of the constraints given by Δ, and distance measures are biased against long-range dependencies.

We tried a number of distance measures, and the one used here encodes the relative position of the heads of the argument and functor (left or right), counts the number of verbs between argument and functor (up to 1), and counts the number of punctuation marks (up to 2). The results are also given in Table 1, and show that, as expected, adding distance gives no improvement overall.

An advantage of the dependency-based evaluation is that results can be given for individual dependency relations. Labelled precision and recall on Section 00 for the most frequent dependency types are shown in Table 2 (for the model without distance measures).[9] The columns # *deps* give the total number of dependencies, first the number put forward by the parser, and second the number in the gold standard. F-score is calculated as (2*LP*LR)/(LP+LR). We also give the scores for the dependencies created by the subject and object relative pronoun categories, including the headless object relative pronoun category.

We would like to compare these results with those of other parsers that have presented dependency-based evaluations. However, the few that exist (Lin, 1995; Carroll et al., 1998; Collins, 1999) have used either different data or different sets of dependencies (or both). In future work we plan to map our CCG dependencies onto the set used by Carroll and Briscoe and parse their evaluation corpus so a direct comparison can be made.

As far as long-range dependencies are concerned, it is similarly hard to give a precise evaluation. Note that the scores in Table 2 currently conflate extracted and in-situ arguments, so that the scores for the direct objects, for example, *include* extracted objects. The scores for the relative pronoun categories give a good indication of the performance on extraction cases, although even here it is not possible at present to determine exactly how well the parser is performing at recovering extracted arguments.

In an attempt to obtain a more thorough analysis, we analysed the performance of the parser on the 24 cases of extracted objects in the gold-standard Section 00 (development set) that were passed down the object relative pronoun category $(NP_X \backslash NP_X)/(S[dcl]/NP_X)$.[10] Of these, 10 (41.7%) were recovered correctly by the parser; 10 were incorrect because the wrong category was assigned to the relative pronoun, 3 were incorrect because the relative pronoun was attached to the wrong noun, and 1 was incorrect because the wrong category was assigned to the predicate from which the object was

[9] Currently all the modifiers in nominal compounds are analysed in CCGbank as N/N, as a default, since the structure of the compound is not present in the Penn Treebank. Thus the scores for N/N are not particularly informative. Removing these relations reduces the overall scores by around 2%. Also, the scores in Table 2 are for around 95% of the sentences in Section 00, because of the problem obtaining gold standard dependency structures for all sentences, noted earlier.

[10] The number of extracted objects need not equal the occurrences of the category since coordination can introduce more than one object per category.

Functor	Relation		LP %	# deps	LR %	# deps	F-score
$N_X/N_{X,1}$	1	*nominal modifier*	92.9	6,769	95.1	6,610	94.0
$NP_X/N_{X,1}$	1	*determiner*	95.7	3,804	95.8	3,800	95.7
$(NP_X\backslash NP_{X,1})/NP_2$	2	*np modifying preposition*	84.2	2,046	77.3	2,230	80.6
$(NP_X\backslash NP_{X,1})/NP_2$	1	*np modifying preposition*	75.8	2,002	74.2	2,045	75.0
$(S_X\backslash NP_Y)\backslash(S_{X,1}\backslash NP_Y)/NP_2$	2	*vp modifying preposition*	60.3	1,368	75.8	1,089	67.2
$(S_X\backslash NP_Y)\backslash(S_{X,1}\backslash NP_Y)/NP_2$	1	*vp modifying preposition*	54.8	1,263	69.4	997	61.2
$(S[dcl]\backslash NP_1)/NP_2)$	1	*transitive verb*	74.8	967	86.4	837	80.2
$(S[dcl]\backslash NP_1)/NP_2)$	2	*transitive verb*	77.4	913	83.6	846	80.4
$(S_X\backslash NP_Y)\backslash(S_{X,1}\backslash NP_Y)$	1	*adverbial modifier*	77.0	683	75.6	696	76.3
(PP/NP_1)	1	*preposition complement*	70.9	729	67.2	769	69.0
$(S[b]\backslash NP_1)/NP_2)$	2	*infinitival transitive verb*	82.1	608	85.4	584	83.7
$(S[dcl]\backslash NP_{X,1})/(S[b]_2\backslash NP_X)$	2	*auxiliary*	98.4	447	97.6	451	98.0
$(S[dcl]\backslash NP_{X,1})/(S[b]_2\backslash NP_X)$	1	*auxiliary*	92.1	455	91.7	457	91.9
$(S[b]\backslash NP_1)/NP_2$	1	*infinitival transitive verb*	79.6	417	78.3	424	78.9
$(NP_X/N_{X,1})\backslash NP_2$	1	*s genitive*	93.2	366	94.5	361	93.8
$(NP_X/N_{X,1})\backslash NP_2$	2	*s genitive*	91.2	365	94.6	352	92.9
$(S[to]_X\backslash NP_{Y,1})/(S[b]_{X,2}\backslash NP_Y))$	1	*to-complementiser*	85.6	320	81.1	338	83.3
$(S[dcl]\backslash NP_1)/S[dcl]_2$	1	*sentential complement verb*	87.1	372	90.0	360	88.5
$(NP_X\backslash NP_{X,1})/(S[dcl]_2\backslash NP_X)$	1	*subject relative pronoun*	73.8	237	69.2	253	71.4
$(NP_X\backslash NP_{X,1})/(S[dcl]_2\backslash NP_X)$	2	*subject relative pronoun*	95.2	229	86.9	251	90.9
$(NP_X\backslash NP_{X,1})/(S[dcl]_2/NP_X)$	1	*object relative pronoun*	66.7	15	45.5	22	54.1
$(NP_X\backslash NP_{X,1})/(S[dcl]_2/NP_X)$	2	*object relative pronoun*	85.7	14	63.2	19	72.8
$NP/(S[dcl]_1/NP)$	1	*headless object relative pronoun*	100.0	10	83.3	12	90.9

Table 2: Results for section 00 by dependency relation

extracted. The tendency for the parser to assign the wrong category to the relative pronoun in part reflects the fact that complementiser *that* is fifteen times as frequent as object relative pronoun *that*. However, the supertagger alone gets 74% of the object relative pronouns correct, if it is used to provide a single category per word, so it seems that our dependency model is further biased against object extractions, possibly because of the technical unsoundness noted earlier.

It should be recalled in judging these figures that they are only a first attempt at recovering these long-range dependencies, which most other wide-coverage parsers make no attempt to recover at all. To get an idea of just how demanding this task is, it is worth looking at an example of object relativization that the parser gets correct. Figure 2 gives part of a dependency structure returned by the parser for a sentence from section 00 (with the relations omitted).[11] Notice that both *respect* and *confidence* are objects of *had*. The relevant dependency quadruples found by the parser are the following:

[11]The full sentence is *The events of April through June damaged the respect and confidence which most Americans previously had for the leaders of China.*

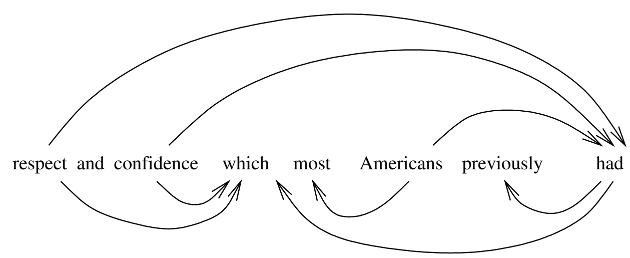


Figure 2: A dependency structure recovered by the parser from unseen data

(13) $\langle \text{which}, (NP_X\backslash NP_{X,1})/(S[dcl]_2/NP_X), 2, \text{had}\rangle$
$\langle \text{which}, (NP_X\backslash NP_{X,1})/(S[dcl]_2/NP_X), 1, \text{confidence}\rangle$
$\langle \text{which}, (NP_X\backslash NP_{X,1})/(S[dcl]_2/NP_X), 1, \text{respect}\rangle$
$\langle \text{had}, (S[dcl]_{had}\backslash NP_1)/NP_2, 2, \text{confidence}\rangle$
$\langle \text{had}, (S[dcl]_{had}\backslash NP_1)/NP_2, 2, \text{respect}\rangle$

7 Conclusions and Further Work

This paper has shown that accurate, efficient wide-coverage parsing is possible with CCG. Along with Hockenmaier and Steedman (2002b), this is the first CCG parsing work that we are aware of in which almost 98% of unseen sentences from the CCGbank can be parsed.

The parser is able to capture a number of long-range dependencies that are not dealt with by existing treebank parsers. Capturing such dependencies is necessary for any parser that aims to support wide-coverage semantic analysis—say to support question-answering in any domain in which the difference between questions like *Which company did Marks sue?* and *Which company sued Marks?* matters. An advantage of our approach is that the recovery of long-range dependencies is fully integrated with the grammar and parser, rather than being relegated to a post-processing phase.

Because of the extreme naivety of the statistical model, these results represent no more than a first attempt at combining wide-coverage CCG parsing with recovery of deep dependencies. However, we believe that the results are promising.

In future work we will present an evaluation which teases out the differences in extracted and in-situ arguments. For the purposes of the statistical modelling, we are also considering building alternative structures that include the long-range dependencies, but which can be modelled using better motivated probability models, such as generative models. This will be important for applying the parser to tasks such as language modelling, for which the possibility of incremental processing of CCG appears particularly attractive.

Acknowledgements

Thanks to Miles Osborne and the ACL-02 referees for comments. Various parts of the research were funded by EPSRC grants GR/M96889 and GR/R02450 and EU (FET) grant MAGICSTER.

References

Steven Abney. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618.

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

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st LREC Conference*, pages 447–454, Granada, Spain.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the NAACL*, pages 132–139, Seattle, WA.

David Chiang. 2000. Statistical parsing with an automatically-extracted Tree Adjoining Grammar. In *Proceedings of the 38th Meeting of the ACL*, pages 456–463, Hong Kong.

Stephen Clark and Julia Hockenmaier. 2002. Evaluating a wide-coverage CCG parser. In *Proceedings of the LREC Beyond PARSEVAL workshop (to appear)*, Las Palmas, Spain.

Stephen Clark. 2002. A supertagger for Combinatory Categorial Grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (to appear)*, Venice, Italy.

Michael Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Meeting of the ACL*, pages 184–191, Santa Cruz, CA.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Jason Eisner. 1996a. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Meeting of the ACL*, pages 79–86, Santa Cruz, CA.

Jason Eisner. 1996b. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th COLING Conference*, pages 340–345, Copenhagen, Denmark.

Julia Hockenmaier and Mark Steedman. 2002a. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the Third LREC Conference (to appear)*, Las Palmas, Spain.

Julia Hockenmaier and Mark Steedman. 2002b. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL (to appear)*, Philadelphia, PA.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic 'unification-based' grammars. In *Proceedings of the 37th Meeting of the ACL*, pages 535–541, University of Maryland, MD.

Nobo Komagata. 1997. Efficient parsing for CCGs with generalized type-raised categories. In *Proceedings of the 5th International Workshop on Parsing Technologies*, pages 135–146, Boston, MA.

Dekang Lin. 1995. A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of IJCAI-95*, pages 1420–1425, Montreal, Canada.

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

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