Hypertagging: Supertagging for Surface Realization with CCG

Dominic Espinosa and **Michael White** and **Dennis Mehay**
Department of Linguistics
The Ohio State University
Columbus, OH, USA
{espinosa,mwhite,mehay}@ling.osu.edu

Abstract

In lexicalized grammatical formalisms, it is possible to separate *lexical category assignment* from the combinatory processes that make use of such categories, such as parsing and realization. We adapt techniques from *supertagging* — a relatively recent technique that performs complex lexical tagging before full parsing (Bangalore and Joshi, 1999; Clark, 2002) — for chart realization in OpenCCG, an open-source NLP toolkit for CCG. We call this approach *hypertagging*, as it operates at a level "above" the syntax, tagging semantic representations with syntactic lexical categories. Our results demonstrate that a hypertagger-informed chart realizer can achieve substantial improvements in realization speed (being approximately twice as fast) with superior realization quality.

1 Introduction

In lexicalized grammatical formalisms such as Lexicalized Tree Adjoining Grammar (Schabes et al., 1988, LTAG), Combinatory Categorial Grammar (Steedman, 2000, CCG) and Head-Driven Phrase-Structure Grammar (Pollard and Sag, 1994, HPSG), it is possible to separate *lexical category assignment* — the assignment of informative syntactic categories to linguistic objects such as words or lexical predicates — from the combinatory processes that make use of such categories — such as parsing and surface realization. One way of performing lexical assignment is simply to hypothesize all possible lexical categories and then search for the best combination thereof, as in the CCG parser in (Hockenmaier, 2003) or the chart realizer in (Carroll and Oepen, 2005). A relatively recent technique for lexical category assignment is *supertagging* (Bangalore and Joshi, 1999), a preprocessing step to parsing that assigns likely categories based on word and part-of-speech (POS) contextual information. Supertagging was dubbed "almost parsing" by these authors, because an oracle supertagger left relatively little work for their parser, while speeding up parse times considerably. Supertagging has been more recently extended to a multitagging paradigm in CCG (Clark, 2002; Curran et al., 2006), leading to extremely efficient parsing with state-of-the-art dependency recovery (Clark and Curran, 2007).

We have adapted this multitagging approach to lexical category assignment for realization using the CCG-based natural language toolkit OpenCCG.[1] Instead of basing category assignment on linear word and POS context, however, we predict lexical categories based on contexts within a directed graph structure representing the logical form (LF) of a proposition to be realized. Assigned categories are instantiated in OpenCCG's chart realizer where, together with a treebank-derived syntactic grammar (Hockenmaier and Steedman, 2007) and a factored language model (Bilmes and Kirchhoff, 2003), they constrain the English word-strings that are chosen to express the LF. We have dubbed this approach *hypertagging*, as it operates at a level "above" the syntax, moving from semantic representations to syntactic categories.

We evaluate this hypertagger in two ways: first,

[1] http://openccg.sourceforge.net.

we evaluate it as a tagger, where the hypertagger achieves high single-best (93.6%) and multitagging labelling accuracies (95.8–99.4% with category per lexical predication ratios ranging from 1.1 to 3.9).[2] Second, we compare a hypertagger-augmented version of OpenCCG's chart realizer with the pre-existing chart realizer (White et al., 2007) that simply instantiates the chart with all possible CCG categories (subject to frequency cutoffs) for each input LF predicate. The hypertagger-seeded realizer runs approximately twice as fast as the pre-existing OpenCCG realizer and finds a larger number of complete realizations, resorting less to chart fragment assembly in order to produce an output within a 15 second per-sentence time limit. Moreover, the overall BLEU (Papineni et al., 2002) and METEOR (Lavie and Agarwal, 2007) scores, as well as numbers of exact string matches (as measured against to the original sentences in the CCGbank) are higher for the hypertagger-seeded realizer than for the pre-existing realizer.

This paper is structured as follows: Section 2 provides background on chart realization in OpenCCG using a corpus-derived grammar. Section 3 describes our hypertagging approach and how it is integrated into the realizer. Section 4 describes our results, followed by related work in Section 5 and our conclusions in Section 6.

2 Background

2.1 Surface Realization with OpenCCG

The OpenCCG surface realizer is based on Steedman's (2000) version of CCG elaborated with Baldridge and Kruijff's multi-modal extensions for lexically specified derivation control (Baldridge, 2002; Baldridge and Kruijff, 2003) and hybrid logic dependency semantics (Baldridge and Kruijff, 2002). OpenCCG implements a symbolic-statistical chart realization algorithm (Kay, 1996; Carroll et al., 1999; White, 2006b) combining (1) a theoretically grounded approach to syntax and semantic composition with (2) factored language models (Bilmes and Kirchhoff, 2003) for making choices among the options left open by the grammar.

In OpenCCG, the search for complete realizations

[2]Note that the multitagger is "correct" if the correct tag is anywhere in the multitag set.

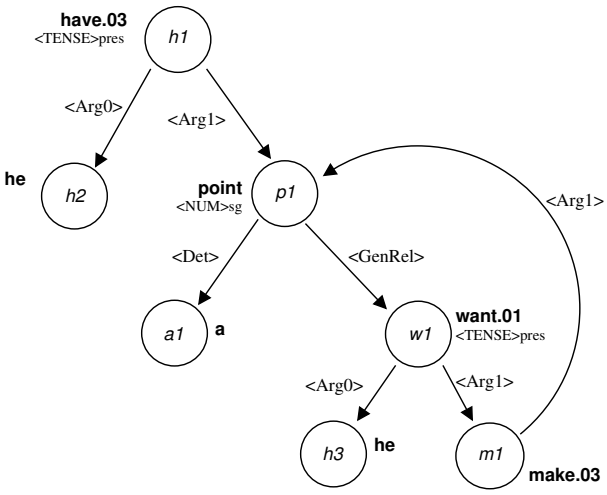

Figure 1: Semantic dependency graph from the CCGbank for *He has a point he wants to make [. . .]*

makes use of n-gram language models over words represented as vectors of factors, including surface form, part of speech, supertag and semantic class. The search proceeds in one of two modes, *anytime* or *two-stage* (packing/unpacking). In the anytime mode, a best-first search is performed with a configurable time limit: the scores assigned by the n-gram model determine the order of the edges on the agenda, and thus have an impact on realization speed. In the two-stage mode, a packed forest of all possible realizations is created in the first stage; in the second stage, the packed representation is unpacked in bottom-up fashion, with scores assigned to the edge for each sign as it is unpacked, much as in (Langkilde, 2000). Edges are grouped into equivalence classes when they have the same syntactic category and cover the same parts of the input logical form. Pruning takes place within equivalence classes of edges. Additionally, to realize a wide range of paraphrases, OpenCCG implements an algorithm for efficiently generating from disjunctive logical forms (White, 2006a).

To illustrate the input to OpenCCG, consider the semantic dependency graph in Figure 1, which is taken from section 00 of a Propbank-enhanced version of the CCGbank (Boxwell and White, 2008). In the graph, each node has a lexical predication (e.g. **make.03**) and a set of semantic features (e.g. ⟨NUM⟩sg); nodes are connected via dependency relations (e.g. ⟨ARG0⟩). Internally, such

graphs are represented using Hybrid Logic Dependency Semantics (HLDS), a dependency-based approach to representing linguistic meaning developed by Baldridge and Kruijff (2002). In HLDS, hybrid logic (Blackburn, 2000) terms are used to describe dependency graphs. These graphs have been suggested as representations for discourse structure, and have their own underlying semantics (White, 2006b).

To more robustly support broad coverage surface realization, OpenCCG has recently been enhanced to greedily assemble fragments in the event that the realizer fails to find a complete realization. The fragment assembly algorithm begins with the edge for the best partial realization, i.e. the one that covers the most elementary predications in the input logical form, with ties broken according to the n-gram score. (Larger fragments are preferred under the assumption that they are more likely to be grammatical.) Next, the chart and agenda are greedily searched for the best edge whose semantic coverage is disjoint from those selected so far; this process repeats until no further edges can be added to the set of selected fragments. In the final step, these fragments are concatenated, again in a greedy fashion, this time according to the n-gram score of the concatenated edges: starting with the original best edge, the fragment whose concatenation on the left or right side yields the highest score is chosen as the one to concatenate next, until all the fragments have been concatenated into a single output.

2.2 Realization from an Enhanced CCGbank

White et al. (2007) describe an ongoing effort to engineer a grammar from the CCGbank (Hockenmaier and Steedman, 2007) — a corpus of CCG derivations derived from the Penn Treebank — suitable for realization with OpenCCG. This process involves converting the corpus to reflect more precise analyses, where feasible, and adding semantic representations to the lexical categories. In the first step, the derivations in the CCGbank are revised to reflect the desired syntactic derivations. Changes to the derivations are necessary to reflect the lexicalized treatment of coordination and punctuation assumed by the multi-modal version of CCG that is implemented in OpenCCG. Further changes are necessary to support semantic dependencies rather than surface syntactic ones; in particular, the features and unification constraints in the categories related to semantically empty function words such complementizers, infinitival-*to*, expletive subjects, and case-marking prepositions are adjusted to reflect their purely syntactic status.

In the second step, a grammar is extracted from the converted CCGbank and augmented with logical forms. Categories and unary type changing rules (corresponding to zero morphemes) are sorted by frequency and extracted if they meet the specified frequency thresholds.

A separate transformation then uses around two dozen generalized templates to add logical forms to the categories, in a fashion reminiscent of (Bos, 2005). The effect of this transformation is illustrated below. Example (1) shows how numbered semantic roles, taken from PropBank (Palmer et al., 2005) when available, are added to the category of an active voice, past tense transitive verb, where ***pred***
is a placeholder for the lexical predicate; examples (2) and (3) show how more specific relations are introduced in the category for determiners and the category for the possessive *'s*, respectively.

(1) $\mathsf{s}_{1:dcl}\backslash\mathsf{np}_{2}/\mathsf{np}_{3} \Longrightarrow$
$\mathsf{s}_{1:dcl,x1}\backslash\mathsf{np}_{2:x2}/\mathsf{np}_{3:x3}\ :\ @_{x1}(\textbf{*pred*} \wedge \langle\textsc{tense}\rangle\text{pres} \wedge \langle\textsc{Arg0}\rangle x2 \wedge \langle\textsc{Arg1}\rangle x3)$

(2) $\mathsf{np}_{1}/\mathsf{n}_{1} \Longrightarrow$
$\mathsf{np}_{1:x1}/\mathsf{n}_{1:x1}\ :\ @_{x1}(\langle\textsc{Det}\rangle(d \wedge \textbf{*pred*}))$

(3) $\mathsf{np}_{1}/\mathsf{n}_{1}\backslash\mathsf{np}_{2} \Longrightarrow$
$\mathsf{np}_{1:x1}/\mathsf{n}_{1:x1}\backslash\mathsf{np}_{2:x2}\ :\ @_{x1}(\langle\textsc{GenOwn}\rangle x2)$

After logical form insertion, the extracted and augmented grammar is loaded and used to parse the sentences in the CCGbank according to the gold-standard derivation. If the derivation can be successfully followed, the parse yields a logical form which is saved along with the corpus sentence in order to later test the realizer. The algorithm for following corpus derivations attempts to continue processing if it encounters a blocked derivation due to sentence-internal punctuation. While punctuation has been partially reanalyzed to use lexical categories, many problem cases remain due to the CCGbank's reliance on punctuation-specific binary rules that are not supported in OpenCCG.

Currently, the algorithm succeeds in creating logical forms for 97.7% of the sentences in the development section (Sect. 00) of the converted CCGbank, and 96.1% of the sentences in the test section (Sect. 23). Of these, 76.6% of the development logical forms are semantic dependency graphs with a single root, while 76.7% of the test logical forms have a single root. The remaining cases, with multiple roots, are missing one or more dependencies required to form a fully connected graph. These missing dependencies usually reflect inadequacies in the current logical form templates.

2.3 Factored Language Models

Following White et al. (2007), we use factored trigram models over words, part-of-speech tags and supertags to score partial and complete realizations. The language models were created using the SRILM toolkit (Stolcke, 2002) on the standard training sections (2–21) of the CCGbank, with sentence-initial words (other than proper names) uncapitalized. While these models are considerably smaller than the ones used in (Langkilde-Geary, 2002; Velldal and Oepen, 2005), the training data does have the advantage of being in the same domain and genre (using larger n-gram models remains for future investigation). The models employ interpolated Kneser-Ney smoothing with the default frequency cutoffs. The best performing model interpolates a word trigram model with a trigram model that chains a POS model with a supertag model, where the POS model conditions on the previous two POS tags, and the supertag model conditions on the previous two POS tags as well as the current one.

Note that the use of supertags in the factored language model to score possible realizations is distinct from the prediction of supertags for lexical category assignment: the former takes the words in the local context into account (as in supertagging for parsing), while the latter takes features of the logical form into account. It is this latter process which we call hypertagging, and to which we now turn.

3 The Approach

3.1 Lexical Smoothing and Search Errors

In White et al.'s (2007) initial investigation of scaling up OpenCCG for broad coverage realization, all categories observed more often than a threshold frequency were instantiated for lexical predicates; for unseen words, a simple smoothing strategy based on the part of speech was employed, assigning the most frequent categories for the POS. This approach turned out to suffer from a large number of search errors, where the realizer failed to find a complete realization before timing out even in cases where the grammar supported one. To confirm that search errors had become a significant issue, White et al. compared the percentage of complete realizations (versus fragmentary ones) with their top scoring model against an oracle model that uses a simplified BLEU score based on the target string, which is useful for regression testing as it guides the best-first search to the reference sentence. The comparison involved both a medium-sized (non-blind) grammar derived from the development section and a large grammar derived from the training sections (the latter with slightly higher thresholds). As shown in Table 1, with the large grammar derived from the training sections, many fewer complete realizations are found (before timing out) using the factored language model than are possible, as indicated by the results of using the oracle model. By contrast, the difference is small with the medium-sized grammar derived from the development section. This result is not surprising when one considers that a large number of common words are observed to have many possible categories.

test set	grammar	complete oracle / best
dev (00)	dev	49.1% / 47.8%
	train	37.5% / 22.6%

Table 1: Percentage of complete realizations using an oracle n-gram model versus the best performing factored language model.

In the next section, we show that a supertagger for CCG realization, or hypertagger, can reduce the problem of search errors by focusing the search space on the most likely lexical categories.

3.2 Maximum Entropy Hypertagging

As supertagging for parsing involves studying a given input word and its local context, the concep-

tual equivalent for a lexical predicate in the LF is to study a given node and its local graph structure. Our implementation makes use of three general types of features: lexicalized features, which are simply the names of the parent and child elementary predication nodes, graph structural features, such as the total number of edges emanating from a node, the number of argument and non-argument dependents, and the names of the relations of the dependent nodes to the parent node, and syntactico-semantic attributes of nodes, such as the tense and number. For example, in the HLDS graph shown in Figure 1, the node representing *want* has two dependents, and the relational type of *make* with respect to *want* is ARG1.

Clark (2002) notes in his parsing experiments that the POS tags of the surrounding words are highly informative. As discussed below, a significant gain in hypertagging accuracy resulted from including features sensitive to the POS tags of a node's parent, the node itself, and all of its arguments and modifiers. Predicting these tags requires the use of a separate POS tagger, which operates in a manner similar to the hypertagger itself, though exploiting a slightly different set of features (e.g., including features corresponding to the four-character prefixes and suffixes of rare logical predication names). Following the (word) supertagging experiments of (Curran et al., 2006) we assigned potentially multiple POS tags to each elementary predication. The POS tags assigned are all those that are some factor β of the highest ranked tag,[3] giving an average of 1.1 POS tags per elementary predication. The values of the corresponding feature functions are the POS tag probabilities according to the POS tagger. At this ambiguity level, the POS tagger is correct $\approx 92\%$ of the time.

Features for the hypertagger were extracted from semantic dependency graphs extracted from sections 2 through 21 of the CCGbank. In total, 37,168 dependency graphs were derived from the corpus, yielding 468,628 feature parameters.

The resulting contextual features and gold-standard supertag for each predication were then used to train a maximum entropy classifier model. Maximum entropy models describe a set of probability distributions of the form:

$$p(o \mid x) = \frac{1}{Z(x)} \cdot exp\Big(\sum_{i=1}^{n} \lambda_i f_i(o, x)\Big)$$

where o is an outcome, x is a context, the f_i are feature functions, the λ_i are the respective weights of the feature functions, and $Z(x)$ is a normalizing sum over all competing outcomes. More concretely, given an elementary predication labeled *want* (as in Figure 1), a feature function over this node could be:

$$f(o, x) = \begin{cases} 1, & \text{if } o \text{ is } (\mathsf{s[dcl]\backslash np})/(\mathsf{s[adj]\backslash np}) \text{ and} \\ & \text{number\_of\_LF\_dependents}(x) = 2 \\ 0, & \text{otherwise.} \end{cases}$$

We used Zhang Le's maximum entropy toolkit[4] for training the hypertagging model, which uses an implementation of Limited-memory BFGS, an approximate quasi-Newton optimization method from the numerical optimization literature (Liu and Nocedal, 1989). Using L-BFGS allowed us to include continuous feature function values where appropriate (e.g., the probabilities of automatically-assigned POS tags). We trained each hypertagging model to 275 iterations and our POS tagging model to 400 iterations. We used no feature frequency cut-offs, but rather employed Gaussian priors with global variances of 100 and 75, respectively, for the hypertagging and POS tagging models.

3.3 Iterative β-Best Realization

During realization, the hypertagger serves to probabilistically filter the categories assigned to an elementary predication, as well as to propose categories for rare or unseen predicates. Given a predication, the tagger returns a β-best list of supertags in order of decreasing probability. Increasing the number of categories returned clearly increases the likelihood that the most-correct supertag is among them, but at a corresponding cost in chart size. Accordingly, the hypertagger begins with a highly restrictive value for β, and backs off to progressively less-restrictive values if no complete realization could be found using the set of supertags returned. The search is restarted

[3] I.e., all tags t whose probabilities $p(t) \geq \beta \cdot p*$, where $p*$ is the highest ranked tag's probability.

[4] `http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html`.

Table 2: Hypertagger accuracy on Sections 00 and 23. Results (in percentages) are for per-logical-predication (PR) and per-whole-graph (GRPH) tagging accurcies. Difference between best-only and baselines (b.l.) is significant ($p < 2 \cdot 10^{-16}$) by McNemar's χ^2 test.

		Sect00		Sect23	
β	$\frac{\text{Tags}}{\text{Pred}}$	PR	GRPH	PR	GRPH
b.l. 1	1	68.7	1.8	68.7	2.3
b.l. 2	2	84.3	9.9	84.4	10.9
1.0	1	93.6	40.4	93.6	38.2
0.16	1.1	95.8	55.7	96.2	56.8
0.05	1.2	96.6	63.8	97.3	66.0
0.0058	1.5	97.9	74.8	98.3	76.9
1.75e-3	1.8	98.4	78.9	98.7	81.8
6.25e-4	2.2	98.7	82.5	99.0	84.3
1.25e-4	3.2	99.0	85.7	99.3	88.5
5.8e-5	3.9	99.1	87.2	99.4	89.9

from scratch with the next β value, though in principle the same chart could be expanded. The iterative, β-best search for a complete realization uses the realizer's packing mode, which can more quickly determine whether a complete realization is possible. If the halfway point of the overall time limit is reached with no complete realization, the search switches to best-first mode, ultimately assembling fragments if no complete realization can be found during the remaining time.

4 Results and Discussion

Several experiments were performed in training and applying the hypertagger. Three different models were created using 1) non-lexicalized features only, 2) all features excluding POS tags, 3) all, 3) all features except syntactico-semantic attributes such as tense and number and 4) all features available. Models trained on these feature subsets were tested against one another on Section 00, and then the best performing model was run on both Section 00 and 23.

4.1 Feature Ablation Testing

The the whole feature set was found in feature ablation testing on the development set to outperform all other feature subsets significantly ($p < 2.2 \cdot 10^{-16}$). These results listed in Table 3. As we can see, taking

Table 3: Hypertagger feature ablation testing results on Section 00. The full feature set outperforms all others significantly ($p < 2.2 \cdot 10^{-16}$). Results for per-predication (PR) and per-whole-graph (GRPH) tagging percentage accuracies are listed. (**Key: no-POS**=no POS features; **no-attr**=no syntactico-semantic attributes such as tense and number; **non-lex**=non-lexicalized features only (no predication names).

FEATURESET	PR	GRPH
full	93.6	40.37
no-POS	91.3	29.5
no-attr	91.8	31.2
non-lex	91.5	28.7

away any one class of features leads to drop in per-predication tagging accuracy of at least 1.8% and a drop per-whole-graph accuracy of at least 9.2%. As expected from previous work in supertagging (for parsing), POS features resulted in a large improvement in overall accuracy (1.8%). Although the POS tagger by itself is only 92% accurate (as a multi-tagger of 1.1 $\frac{POS}{word}$ average ambiguity) — well below the state-of-the-art for the tagging of words — its predictions are still quite valuable to the hypertagger.

4.2 Best Model Hypertagger Accuracy

The results for the full feature set on Sections 00 and 23 are outlined in Table 2. Included in this table are accuracy data for a baseline dummy tagger which simply assigns the most-frequently-seen tag(s) for a given predication and backs off to the overall most frequent tag(s) when confronted with an unseen predication. The development set (00) was used to tune the β parameter to obtain reasonable hypertag ambiguity levels; the model was not otherwise tuned to it. The hypertagger achieves high per-predication and whole-graph accuracies even at small ambiguity levels.

4.3 Realizer Performance

Tables 4 and 5 show how the hypertagger improves realization performance on the development and test sections of the CCGbank. As Table 4 indicates, using the hypertagger in an iterative beta-best fashion more than doubles the number of grammatically complete realizations found within the time

Table 5: Realization quality metrics exact match, BLEU and METEOR, on complete realizations only and overall, with and without hypertagger, on Sections 00 and 23.

Section	Hypertagger	Complete		Overall		
		BLEU	METEOR	Exact	BLEU	METEOR
00	with	0.8137	0.9153	15.3%	0.6567	0.8494
	w/o	0.6864	0.8585	11.3%	0.5902	0.8209
23	with	**0.8149**	0.9162	16.0%	**0.6701**	0.8557
	w/o	0.6910	0.8606	12.3%	0.6022	0.8273

Table 4: Percentage of grammatically complete realizations, runtimes for complete realizations and overall runtimes, with and without hypertagger, on Sections 00 and 23.

Section	Hypertagger	Percent Complete	Complete Time	Overall Time
00	with	47.4%	1.2s	4.5s
	w/o	22.6%	8.7s	9.5s
23	with	48.5%	**1.2s**	**4.4s**
	w/o	23.5%	8.9s	9.6s

limit; on the development set, this improvement elimates more than the number of known search errors (cf. Table 1). Additionally, by reducing the search space, the hypertagger cuts overall realization times by more than half, and in the cases where complete realizations are found, realization times are reduced by a factor of four, down to 1.2 seconds per sentence on a desktop Linux PC.

Table 5 shows that increasing the number of complete realizations also yields improved BLEU and METEOR scores, as well as more exact matches. In particular, the hypertagger makes possible a more than 6-point improvement in the overall BLEU score on both the development and test sections, and a more than 12-point improvement on the sentences with complete realizations.

As the effort to engineer a grammar suitable for realization from the CCGbank proceeds in parallel to our work on hypertagging, we expect the hypertagger-seeded realizer to continue to improve, since a more complete and precise extracted grammar should enable more complete realizations to be found, and richer semantic representations should simplify the hypertagging task. Even with the current incomplete set of semantic templates, the hypertagger brings realizer performance roughly up to state-of-the-art levels, as our overall test set BLEU score (0.6701) slightly exceeds that of Cahill and van Genabith (2006), though at a coverage of 96% instead of 98%. We caution, however, that it remains unclear how meaningful it is to directly compare these scores when the realizer inputs vary considerably in their specificity, as Langkilde-Geary's (2002) experiments dramatically illustrate.

5 Related Work

Our approach follows Langkilde-Geary (2002) and Callaway (2003) in aiming to leverage the Penn Treebank to develop a broad-coverage surface realizer for English. However, while these earlier, generation-only approaches made use of converters for transforming the outputs of Treebank parsers to inputs for realization, our approach instead employs a shared bidirectional grammar, so that the input to realization is guaranteed to be the same logical form constructed by the parser. In this regard, our approach is more similar to the ones pursued more recently by Carroll, Oepen and Velldal (2005; 2005; 2006), Nakanishi et al. (2005) and Cahill and van Genabith (2006) with HPSG and LFG grammars.

While we consider our approach to be the first to employ a supertagger for realization, or hypertagger, the approach is clearly reminiscent of the LTAG tree models of Srinivas and Rambow (2000). The main difference between the approaches is that ours consists of a multitagging step followed by the bottom-up construction of a realization chart, while theirs involves the top-down selection of the single most likely supertag for each node that is grammatically

compatible with the parent node, with the probability conditioned only on the child nodes. Note that although their approach does involve a subsequent lattice construction step, it requires making non-standard assumptions about the TAG; in contrast, ours follows the chart realization tradition of working with the same operations of grammatical combination as in parsing, including a well-defined notion of semantic composition. Additionally, as our tagger employs maximum entropy modeling, it is able to take into account a greater variety of contextual features, including those derived from parent nodes.

In comparison to other recent chart realization approaches, Nakanishi et al.'s is similar to ours in that it employs an iterative beam search, dynamically changing the beam size in order to cope with the large search space. However, their log-linear selection models have been adapted from ones used in parsing, and do not condition choices based on features of the input semantics to the same extent. In particular, while they employ a baseline maximum likelihood model that conditions the probability of a lexical entry upon its predicate argument structure (PAS) — that is, the set of elementary predications introduced by the lexical item — this probability does not take into account other elements of the local context, including parents and modifiers, and their lexical predicates. Similarly, Cahill and van Genabith condition the probability of their lexical rules on the set of feature-value pairs linked to the RHS of the rule, but do not take into account any additional context. Since their probabilistic models involve independence assumptions like those in a PCFG, and since they do not employ n-grams for scoring alternative realizations, their approach only keeps the single most likely edge in an equivalence class, rather than packing them into a forest. Carroll, Oepen and Velldal's approach is like Nakanishi et al.'s in that they adapt log-linear parsing models to the realization task; however, they employ manually written grammars on much smaller corpora, and perhaps for this reason they have not faced the need to employ an iterative beam search.

6 Conclusion

We have introduced a novel type of supertagger, which we have dubbed a *hypertagger*, that assigns CCG category labels to elementary predications in a structured semantic representation with high accuracy at several levels of tagging ambiguity in a fashion reminiscent of (Bangalore and Rambow, 2000). To our knowledge, we are the first to report tagging results in the semantic-to-syntactic direction. We have also shown that, by integrating this hypertagger with a broad-coverage CCG chart realizer, considerably faster realization times are possible (approximately twice as fast as compared with a realizer that performs simple lexical look-ups) with higher BLEU, METEOR and exact string match scores. Moreover, the hypertagger-augmented realizer finds more than twice the number of complete realizations, and further analysis revealed that the realization quality (as *per* modified BLEU and METEOR) is higher in the cases when the realizer finds a complete realization. This suggests that further improvements to the hypertagger will lead to more complete realizations, hence more high-quality realizations. Finally, further efforts to engineer a grammar suitable for realization from the CCGbank should provide richer feature sets, which, as our feature ablation study suggests, are useful for boosting hypertagging performance, hence for finding better and more complete realizations.

Acknowledgements

The authors thank the anonymous reviewers, Chris Brew, Detmar Meurers and Eric Fosler-Lussier for helpful comments and discussion.

References

Jason Baldridge and Geert-Jan Kruijff. 2002. Coupling CCG and Hybrid Logic Dependency Semantics. In *Proc. ACL-02*.

Jason Baldridge and Geert-Jan Kruijff. 2003. Multi-Modal Combinatory Categorial Grammar. In *Proc. ACL-03*.

Jason Baldridge. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, School of Informatics, University of Edinburgh.

Srinivas Bangalore and Aravind K. Joshi. 1999. Su-

pertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265.

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proce. COLING-00*.

Jeff Bilmes and Katrin Kirchhoff. 2003. Factored language models and general parallelized backoff. In *Proc. HLT-03*.

Patrick Blackburn. 2000. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–625.

Johan Bos. 2005. Towards wide-coverage semantic interpretation. In *Proc. IWCS-6*.

Stephen Boxwell and Michael White. 2008. Projecting Propbank roles onto the CCGbank. In *Proc. LREC-08*. To appear.

Aoife Cahill and Josef van Genabith. 2006. Robust PCFG-based generation using automatically acquired LFG approximations. In *Proc. COLING-ACL '06*.

Charles Callaway. 2003. Evaluating coverage for large symbolic NLG grammars. In *Proc. IJCAI-03*.

John Carroll and Stefan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *Proc. IJCNLP-05*.

John Carroll, Ann Copestake, Dan Flickinger, and Victor Poznański. 1999. An efficient chart generator for (semi-) lexicalist grammars. In *Proc. ENLG-99*.

Stephen Clark and James Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4).

Stephen Clark. 2002. Supertagging for combinatory categorial grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pages 19–24, Venice, Italy.

James R. Curran, Stephen Clark, and David Vadas. 2006. Multi-tagging for lexicalized-grammar parsing. In *Proceedings of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING/ACL-06)*, pages 697–704, Sydney, Australia.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

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

Martin Kay. 1996. Chart generation. In *Proc. ACL-96*.

Irene Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proc. INLG-02*.

Irene Langkilde. 2000. Forest-based statistical sentence generation. In *Proc. NAACL-00*.

Alon Lavie and Abhaya Agarwal. 2007. METEOR: An automatic metric for MT evaluation with high levels of correlation with human judgments. In *Proceedings of Workshop on Statistical Machine Translation at the 45th Annual Meeting of the Association of Computational Linguistics (ACL-2007)*, Prague.

D C Liu and Jorge Nocedal. 1989. On the limited memory method for large scale optimization. *Mathematical Programming B*, 45(3).

Hiroko Nakanishi, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic methods for disambiguation of an HPSG-based chart generator. In *Proc. IWPT-05*.

Martha Palmer, Dan Gildea, and Paul Kingsbury. 2005. The proposition bank: A corpus annotated with semantic roles. *Computational Linguistics*, 31(1).

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA.

Carl J Pollard and Ivan A Sag. 1994. *Head-Driven Phrase Structure Grammar*. University Of Chicago Press.

Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88)*, Budapest.

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

Andreas Stolcke. 2002. SRILM — An extensible language modeling toolkit. In *Proc. ICSLP-02*.

Erik Velldal and Stephan Oepen. 2005. Maximum entropy models for realization ranking. In *Proc. MT Summit X*.

Erik Velldal and Stephan Oepen. 2006. Statistical ranking in tactical generation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, July.

Michael White, Rajakrishnan Rajkumar, and Scott Martin. 2007. Towards broad coverage surface realization with CCG. In *Proc. of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT)*.

Michael White. 2006a. CCG chart realization from disjunctive inputs. In *Proceedings, INLG 2006*.

Michael White. 2006b. Efficient realization of coordinate structures in Combinatory Categorial Grammar. *Research on Language and Computation*, 4(1):39–75.